BLOOM FILTER-BASED CONTENT DISCOVERY AND RETRIEVAL FOR INFORMATION-CENTRIC NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Sayed Ali Marandi

von Iran

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik

BLOOM FILTER-BASED CONTENT DISCOVERY AND RETRIEVAL FOR INFORMATION-CENTRIC NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Sayed Ali Marandi

von Iran

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, November, 2020

Der Dekan:
Prof. Dr. Zoltan Balogh

Dedicated to my mother…

Acknowledgements

During my doctoral studies, several people have given me guidance and support. In the following, I would like to gratefully acknowledge their guidance and support.
I would like to thank my thesis director, *Prof. Dr. Torsten Braun*, for the opportunity to pursue my studies towards a Ph.D. at the Communications and Distributed Systems lab (CDS) of the University of Bern. I express my sincere gratitude to you for the advice, feedback, and corrections that improved the quality of my work and papers until the completion of this thesis.
I would like to express my deepest gratitude to *Prof. Dr. Nikolaos Thomos (Nikos)*, for all the discussions, support, ideas, and motivation he has given me. Nikos, I would like to thank you very much for all the fruitful discussions, for all the detailed comments and corrections that improved the quality of my work.
I would like to also express my deep and sincere gratitude to *Prof. Dr. Kavé Salamatian (Kavé)*. Thank you very much Kavé, for all the discussions, and for all that I have learned from you over the years.
I gratefully thank Mrs. *Daniela Schroth* for all her support. Thank you very much Daniela, for your kind help and support at the beginning of my doctoral studies that I needed a lot of information to settle in Switzerland, and during my work for all the administrative letters, and your help and support when I came to the office asking questions or having requests.
I would like to thank my colleagues at CDS, for the good time we spent together and for all the discussions and support.
I gratefully thank my family for all the unconditional love and support they gave me. Although they were far from me, their thoughts and wishes were always with me.

Bern, 1 September 2019 A. Marandi

Abstract

Named Data Networking (NDN) requires routing protocols that use content object names for routing clients' requests. In this thesis, we develop routing protocols for NDN based on content advertisements that we compress using Bloom filters (BF). We propose *push-based Bloom Filter-based Routing (BFR)* and *pull-based BFR* for NDN. Push-based BFR advertises all provided content object names, while pull-based BFR only advertises the requested content object names. Therefore, pull-based BFR outperforms push-based BFR in terms of the required communication and storage overhead for content advertisements. To reduce content retrieval delay, we propose to use Network Coding (NC)-based content retrieval. We use the BF-based information distributed for content discovery to select network codes. The proposed NC-based protocol uses a constraint on the equation system size and BF-based feedbacks to control codeblock size. We show that the proposed NC-based protocol achieves lower average content block retrieval delay than push-based and pull-based BFR. Service-Centric Networking [21] requires load balancing mechanisms to route service requests. To address this requirement, Layered-Service Centric Networking (L-SCN) [31] proposed to divide nodes into domains where each domain of nodes is managed by a supernode. However, L-SCN lacks algorithms to select supernodes in the network topology. We present supernode selection algorithms based on the construction of Dominating Sets (DS) and Connected Dominating Sets (CDS) over the network topology. Then, we propose intra-domain and inter-domain BF-based routing protocols for routing service requests. We show that our CDS-based routing protocols require much less bandwidth overhead for routing than both DS-based routing and NDN multicast strategy. Further, we show that both DS-based and CDS-based routing protocols achieve significantly less service retrieval delay than the NDN multicast strategy.

Keywords: Named Data Networking, Routing, Bloom Filters, Network Coding, Service-Centric Networking, Dominating Sets.

Contents

List of Figures

1 Introduction

In this chapter, we introduce Information-Centric Networking (ICN) [11]. Then, we discuss the research questions and the thesis contributions to answer the research questions. Finally, we outline the thesis structure.

1.1 Information-Centric Networking

Content retrieval is location-dependent in the current Internet. When a client requests a content object, the request has to reach the server that stores the requested content object. This means that content objects are coupled with the locations of the servers that are identified by IP addresses. With the growth of content objects, the Internet has become a content distribution network. In a content distribution network, clients do not care about the location of the requested content objects. Rather, they care about retrieving them with low delay and with satisfactory quality when the communication involves multimedia services. Therefore, the location-dependent content retrieval in the current Internet does not match with the clients' perspective. Furthermore, location-dependent content retrieval does not use the storage resources provided by

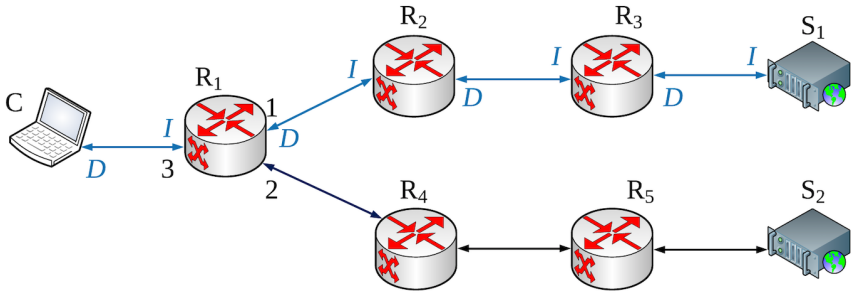


Figure 1.1: A topology to describe NDN

intermediate routers, increases the load on servers, and creates congestion towards servers. To cope with the shortcomings of the location-dependent content retrieval of the Internet, ICN [11] has been proposed. Different ICN projects led to a variety of ICN architectures [37, 41, 73, 88], which are designed based on the following principles: 1) named content objects, 2) in-network caching, and 3) content-based security. In this thesis, we focus on Named Data Networking (NDN) [88] as it is one of the most prominent ICN architectures.

In NDN, hierarchical names are assigned to content objects. For instance, the name */cds.unibe.ch/publications/theses/marandi.pdf* describes the hierarchical name for the PDF file of this thesis cached at the University of Bern's repository. Further, the name of a content object might describe its attributes, e.g., the type of the content object (video, image, etc.), the quality, sequence number, etc.

NDN classifies nodes into the following three types: 1) clients, 2) servers, and 3) routers. In the network topology in Fig. 1.1, there is a client C, five routers R_1, R_2, R_3, R_4, R_5, and two servers S_1, S_2. The links that connect network nodes are called *faces*. For example, in Fig. 1.1, router R_1 is connected to routers R_2 and R_3 through faces 1 and 2, respectively.

NDN defines two types of messages, namely *Interest* and *Data* messages. NDN divides each content object into many segments, which are identified by sequence numbers. To retrieve each segment, a client requires to send an Interest message. Each node maintains the following tables: 1) Pending Interest Table (PIT), 2) Content Store (CS), and 3) Forwarding Information Base (FIB). PITs store the received Interest messages and keep records of the faces over which Interest messages have been received or forwarded, CSs store the received Data messages, and FIBs store the next hop faces for different name prefixes. To describe Interest message forwarding and

PIT of R_1	
Name	/cds.unibe.ch/publications/theses/marandi.pdf/1
Incoming face(s)	3
Outgoing face(s)	1

Figure 1.2: PIT of router R_1

FIB of R_1	
Name prefix	/cds.unibe.ch/publications/theses/
Next hop face(s)	1

Figure 1.3: FIB of router R_1

Data message retrieval, in Fig. 1.1 we assume that client C requires to retrieve a Data message with name $/cds.unibe.ch/publications/theses/marandi.pdf/1$, which is the first segment for the file of this thesis. Client C has to send an Interest message I with the above name. Since client C is connected to router R_1 via a single face, client C forwards Interest message I to router R_1. Router R_1 receives Interest message I over face 3 and stores this Interest in its PIT, which is shown in Fig. 1.2. Then, router R_1 has to decide over which face(s), i.e., face 1, face 2, or both, to forward Interest message I. To make this decision, router R_1 has to look up its FIB table, which is shown in Fig. 1.3, to decide the next hop face over which Interest message I has to be forwarded.

NDN performs longest prefix match operations on Interest names for FIB lookup operations. As Fig. 1.3 shows, the FIB table of router R_1 indicates that the next hop face for name prefix $/cds.unibe.ch/publications/theses/$ is face 1. Thus, router R_1 forwards Interest message I over face 1. Then router R_2 receives Interest message I, checks its FIB table, and forwards Interest message I to router R_3, which finally forwards Interest message I to server S_1 that stores the Data message for name $/cds.unibe.ch/publications/theses/marandi.pdf/1$.

To reply to Interest message I, server S_1 will return a Data message D to router R_3, router R_3 will forward the Data message D to router R_2, R_2 will forward D to R_1, and eventually R_1 will deliver D to client C that initially requested it. From this example, we learn that Data messages travel over the reverse paths of their corresponding Interest messages.

When Data messages pass by routers, those might leave a copy of them in their CS tables. This feature of NDN is called in-network caching. For the above example, Fig.

CS of R_1	
Name prefix	/cds.unibe.ch/publications/theses/marandi.pdf/1
Data payload	…

Figure 1.4: CS of router R_1

1.4 shows the CS of router R_1. In-network caching is one of the advantages of NDN over IP networks. In IP networks, if a client wants to request a content object, it has to forward a request message for the requested content object up to the content provider server. Since many clients request popular content objects for which the requests have to always reach servers, servers might be overloaded and high traffic towards them might lead to congestion that increases the content retrieval delay and degrades the quality of experience for users. In contrast, NDN allows in-network caching. As a result, content objects will be cached closer to the clients. Further, in-network caching leads to the support of multicast communications.

Another advantage of NDN is content-centric security [88]. IP systems rely on the location of content objects to provide an end-to-end secure tunnel between the requester and the server. On the contrary, NDN proposes to design location-independent security mechanisms by securing the content objects rather than the paths over which they are communicated. Therefore, NDN mandates the servers to cryptographically sign their Data messages for authenticity. Further, NDN permits encryption in case Data messages require to be confidential [90]. Therefore, NDN creates Data authenticity and Data confidentiality regardless of servers' locations and regardless of communication paths [90]. In this thesis, we investigate routing and content retrieval in NDN. Therefore, the research problems that are related to NDN security are beyond the scope of this thesis.

Nowadays, users not only request content objects but they might request computed content objects such as transcoded audio/video files, analyzed images, Google map directions, etc. Such computed content objects are called services [21]. The work in [21] proposed Service-Centric Networking (SCN) to enhance ICN so that it supports services. In SCN, provided services are functions, e.g., transcode a video. Service providers run the software for service functions. The existing implementation of SCN is based on NDN architecture. Similar to NDN, SCN clients use Interest messages to request services. When a service request reaches a service provider that can provide the demanded service, the service provider runs a function to calculate the service response, and, then places the service response in a Data message that is returned,

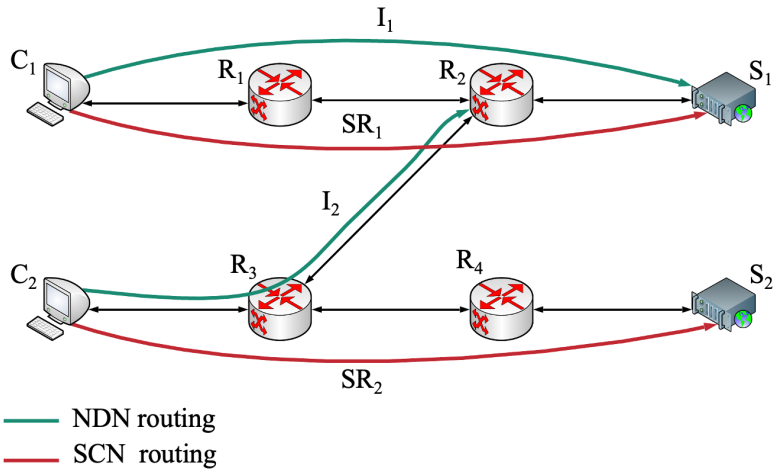


Figure 1.5: A topology to describe routing in SCN

over the reverse path of the service request, to the client.

To route service requests, SCN routing protocols require to discover the services that servers provide and servers' available resources (i.e., CPU, GPU, RAM). NDN routing protocols forward Interest messages with the same names towards the same origin servers to benefit from in-network caching. However, SCN routing protocols require to balance the computational load among different servers. We use Fig. 1.5 to explain the difference between routing in NDN and SCN with an example. In Fig. 1.5, we consider the following two cases: 1) NDN routing, 2) SCN routing. In case 1), we assume that client C_1 sends an Interest message I_1 to request a content object from server S_1. Server S_1 will send a Data message in response to the Interest message I_1, which will be cached at router R_2. If client C_2 issues an Interest message I_2 for the same content object later, router R_3 will forward Interest message I_2 towards server S_1 again, and Interest message I_2 will be satisfied at router R_2 where holds a copy of the requested content object. In case 2), we consider that client C_1 sends a service request SR_1 towards server S_1, and, later, client C_2 issues another service request SR_2. In this case, it is better to forward SR_2 towards server S_2 provided that server S_1 is busy serving client C_1's service request to balance the computational load between servers S_1 and S_2.

Layered-Service Centric Networking (L-SCN) [31] is proposed as a routing architecture for SCN that aims to provide service discovery, resource discovery, and load balancing mechanisms. For scalability reasons, L-SCN suggests clustering network nodes into domains. Each domain is managed by one or more supernodes, which have significant

knowledge about the available services and resources in the domain. Before service routing, L-SCN assumes that supernodes are already selected and the network nodes are already clustered. Therefore, L-SCN lacks algorithms to select supernodes and to cluster network nodes.

1.2 Research Questions

In the following, we pose the Research Questions (RQ) of this thesis.

RQ 1: How to route Interest messages in NDN?

RQ 2: How to reduce bandwidth and storage overhead of Bloom filter-based content advertisements?

RQ 3: How to reduce content retrieval delay in NDN?

RQ 4: How to select supernodes in L-SCN?

1.3 Thesis Contributions

According to the RQs, in the following, we describe the thesis contributions.

1.3.1 Push-based Bloom Filter-based Routing for Named Data Networking

In Chapter 3, we answer RQ 1 by presenting push-based Bloom Filter-based Routing (BFR) as a routing protocol for NDN. To route Interest messages, routing protocols require to locate content objects to populate FIBs. If servers advertise the names of their available content objects, clients and routers will be aware of the provided name prefixes and the paths to reach them. Nevertheless, content objects have long hierarchical names. For example, the average Uniform Resource Locators (URLs) size in the realistic HTTP request database presented in [26] is 42.5 bytes. Therefore, if servers decide to advertise content object names using regular arrays, it will consume significant bandwidth and storage resources. To cope with this problem, servers require to compress content advertisements so that they do not entail significant bandwidth and storage overhead.

To compress content advertisements, in Chapter 3, we argue that it is very promising to use Bloom Filters (BFs). A BF is a well-known data structure for compact set representation. A BF has a very simple structure consisting of a bit vector with size m and k hash functions. There are two advantages of using a BF for representing a set than a regular array: 1) compressed representation of the set, and 2) less complex element search. The complexity of searching an element in a BF is $O(1)$, whereas the complexity of searching an element in a regular array with size n is $O(n)$. BFs have been used in IP networking for different purposes, e.g., finding Longest Prefix Match, probabilistic routing algorithms, summary cache exchange, and matching IP addresses [22, 54, 55]. In NDN, BFs have been used for similar purposes with IP systems [48, 59, 60].

Motivated by the above advantages of using BFs for content advertisements, in Chapter 3 we investigate routing based on BF-based content advertisements. Based on this research, we present push-based BFR [50] as a fully distributed routing protocol for NDN. Servers frequently represent and advertise the names of their provided content objects using BFs. BF-based content advertisements significantly reduce the required bandwidth for content advertisements. When clients and routers receive content advertisements transmitted by the servers, they store them in the PIT tables and use them for routing Interest messages. Push-based BFR does not rely on any IP-based mechanism for routing, and, therefore, it is a fully content-oriented routing protocol. Moreover, push-based BFR does not require any information about the network topology for routing.

We compared push-based BFR with flooding, shortest path, and COBRA [72] routing protocols. Push-based BFR does not require additional protocols that demand high bandwidth resources to calculate the shortest paths. We observed that push-based BFR uses significantly less bandwidth resources for content advertisements compared to the bandwidth resources used for calculating the shortest paths by shortest path routing. Further, push-based BFR does not flood Interest messages. Our results made clear that push-based BFR outperforms flooding and COBRA in terms of the required bandwidth resources for routing and content retrieval, and the average round-trip delay. We compared the performance of push-based BFR with COBRA, and we saw from the results that using push-based BFR, network nodes require much less memory for storing routing information. Therefore, for scenarios that nodes have restricted memory (e.g., IoT scenarios), push-based BFR is a more appropriate routing protocol.

1.3.2 Pull-based Bloom Filter-based Routing for Named Data Networking

In Chapter 4, we answer RQ 2 by presenting pull-based BFR. Although push-based BFR compresses content advertisements using BFs, in this protocol the servers advertise all of their provided content objects. Therefore, the required bandwidth and storage overhead of push-based BFR linearly increases with the content universe size, i.e., the total number of provided content objects. Thus, when the content universe is large, advertising all the provided content objects demands considerable bandwidth resources. Clients only request a small number of all content objects from the entire content universe. Therefore, the proposed pull-based BFR protocol suggests that servers only advertise the demanded content objects. Our results make clear that pull-based BFR requires significantly less bandwidth and storage resources for propagating and storing content advertisements. Further, pull-based BFR also achieves better delay results than push-based BFR when there are restricted link capacities. Moreover, we observed from the results that pull-based BFR is more robust to BF false positive reports than push-based BFR [51].

1.3.3 Network Coding-based Content Retrieval based on Bloom Filter-based Content Discovery

To answer RQ 3, in Chapter 5, we propose to use network coding [12] to reduce content retrieval delay. For routing, we use a BF-based pull method similar to our previous work titled “pull-based BFR” [51] in which clients map the hashed values of Interest message names into Interest BFs and routers aggregate the Interest BFs similar to the pull-based BFR routing protocol. When servers receive the Interest BFs, they do not send content advertisement messages. Rather, servers select linear combinations of the Data messages that are requested using Interest BFs and return these linear combinations via network coded Data messages over the reverse path of the Interest BFs.

When network coding is used, it is important to restrict the number of combined variables. Otherwise, the nodes end up having equation systems in which the number of variables is higher than the number of equations. In such a situation, nodes cannot decode the stored network coded messages. To address this problem, we permit each node to define a capacity constraint, i.e., the number of new variables that

a node can accept in its equation system. Each node also has to signal the set of variables that are already involved in its equation system. If a node has already decoded some variables, it will prefer not to receive linear combinations that are composed of those decoded variables. Hence, nodes can signal the sets of their already decoded variables. We propose that each node frequently sends local feedback messages containing its capacity constraint, a decodingBF representing the set of variables involved in the equation system, and a decodedBF consisting of the variables that the node has already decoded. Servers and routers use the information stored in these local feedback messages to select network codes that do not violate the capacity constraint of their neighbors. We compared the proposed network coding-based protocol with push-based and pull-based BFR. The results made clear that the proposed network coding-based protocol outperforms push-based and pull-based BFR in terms of the required bandwidth resources for content discovery and average content block retrieval delay [52].

1.3.4 Bloom Filter-based Routing for Dominating Set-based Service-Centric Networking

To answer RQ 4, in Chapter 6, we propose to benefit from DS and CDS concepts [44] to select supernodes. Therefore, in Chapter 6, we present distributed algorithms to construct DSs and CDSs over an arbitrary network topology. Next, supernodes implement BF-based pulling of the services available inside the domain; supernodes also solicit information about the available resources (CPU, GPU, RAM). Finally, we have proposed intra-domain and inter-domain routing algorithms for both DS-based and CDS-based clustered networks to route service requests inside and between the domains.

We implemented our clustering and routing algorithms over GEANT network topology [2] as well as three instances of Rocketfuel topology [7] with different sizes. The results show that the bandwidth required for constructing a DS or a CDS increases with the topology size. Further, we compare the performance of the proposed DS and CDS-based routing protocols with the NDN multicast strategy. The results make clear that for large topologies, CDS-based routing entails significantly less bandwidth overhead for routing service requests than both DS-based routing and NDN multicast strategies. Finally, from the results, we see that the proposed DS and CDS-based routing protocols achieve much less service retrieval time than the NDN multicast

strategy [53].

1.4 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 provides an overview of the state-of-the-art on ICN architectures, routing protocols for ICN, BFs, and network coding. Chapter 3 presents the proposed push-based BFR as a routing protocol for NDN. Chapter 4 discusses the scalability issues of push-based BFR and proposes pull-based BFR protocol that addresses those challenges. Chapter 5 describes a network coding-based protocol that makes use of the information disseminated for BF-based content discovery to select network codes for improving content retrieval delay. Chapter 6 describes BF-based routing protocols for (C)DS-based SCNs. Finally, Chapter 7 concludes the thesis and presents some future research directions.

2

State of the Art

2.1 Overview

NDN [88] is the most prominent ICN-based architecture. A large research community is conducting research based on NDN using its open-source implementation. In the Introduction, we posed the RQs related to NDN routing, network coding-based content retrieval in NDN, and SCN routing. In this Chapter, we discuss the most relevant previous works that relate to the RQs and the concepts that we use in the rest of the thesis. We begin by introducing ICN and its more popular architectural proposals. Next, we discuss the related works on NDN routing protocols. Then, we discuss network coding and network coding-based ICN schemes. Finally, we briefly discuss the research works that relate to SCN.

2.2 Information-Centric Networking

Different ICN architectures have been proposed based on different perspectives, for example Data-Oriented Network Architecture (DONA) [41], Publish Subscribe Internet

Technology (PURSUIT) [74], Network of Information (NetInf) [28], Mobility First [3], Content-Centric Networking (CCN) [1], and NDN [4]. These architectures use content names instead of endpoint identifiers but they differ from each other in naming and content discovery protocols [16]. In the following, we briefly describe the above ICN-based architectures. A more detailed description of the proposed ICN-based structures is available in [11] and [85].

2.2.1 Data-Oriented Network Architecture

Data-Oriented Network Architecture (DONA) [41] was proposed as one of the first architectures for ICN. DONA uses flat names in the form of $P : L$, where P is the hashed value of the public key possessed by the content provider, and L is a unique ID of one of the content objects provided by the same content provider. To perform name resolution in DONA, some of the routers are appointed as Resolution Handlers (RH), which are connected to each other for inter-domain routing. RHs distribute information about new content objects registered by the local content providers so that the other RHs know where to find those content objects. When a client wants to request a content object, it sends a *Find* message to the RH it is associated with. The associated RH then forwards the client's request to the RH that is connected to the content provider. Content objects can take the same route that their corresponding *Find* message has traveled, or they can take a different route. RHs might cache the content objects that pass through them while there are on their way to reach the clients.

2.2.2 Publish-Subscribe Internet Technology

Publish Subscribe Internet Technology (PURSUIT) [74] replaces the IP protocol stack with a publish/subscribe protocol stack. PURSUIT architecture has three main components: 1) *rendezvous*, 2) *topology manager*, and 3) *forwarding*. The rendezvous function is responsible for connecting subscribers to publishers. For name resolution, a series of rendezvous nodes form a hierarchical Distributed Hash Table (DHT). When a publisher is identified, rendezvous nodes ask the topology manager to specify a route that connects the publisher and the subscriber to deliver the requested content object. The topology manager uses BFs to compactly represent the routes that are written into message headers. Some forwarding nodes read the routes encoded in message headers and forward the requested content object over the specified route

until the requested content object is delivered to the subscriber. The naming style in PURSUIT follows a *scopeID:rendezvousID* pattern. The role of scopeID is to keep items of related information together, while rendezvousID identifies the content object.

2.2.3 Network of Information

Network of Information (NetInf) [10] is an architecture that inherits features from both PURSUIT [74] and NDN [88]. NetInf has a hierarchical naming style, where Longest Prefix Matching (LPM) is performed for routing operations (similar to NDN). Nevertheless, to bind a subscription to a publication, it requires to have an exact name matching (similar to PURSUIT). NetInf has a hybrid name resolution approach. That is, first a multi-level Distributed Hash Table (DHT) is used to acquire information about the publisher locator, and to route the request to an area, where more information is available about the publisher locator. After, content routers perform name-based routing until the request reaches its destination. In NetInf, content providers announce their provided content objects using a PUBLISH message. These messages are used to route the requests towards content providers. Clients use GET messages to solicit content objects. Apart from GET messages, clients can also send SEARCH messages containing keywords to receive names of some content objects that match the request as well as the location that the matching content objects are stored.

2.2.4 MobilityFirst

MobilityFirst [3] is an ICN project that focuses on users' mobility. MobilityFirst identifies all users, devices, and content objects using unique 160-bits flat IDs. These unique IDs do not depend on network addresses. Nevertheless, it is possible to translate each unique ID to one or several network addresses. This is useful to be able to re-route the messages according to the mobility patterns of devices or content objects. To find the current location of a unique ID, one can contact the name resolution service. In MobilityFirst, the role of the name resolution service is to map the unique IDs to network addresses. Thus, messages are routed according to network addresses.

2.2.5 Content-Centric Networking

Content-Centric Networking (CCN) [37] was proposed as a fully content-oriented and fully distributed ICN architecture. CCN identifies content objects with *names*. When a client wants to request a content object, it issues an *Interest* message with the name of the requested content object. An Interest message travels in the network until it reaches a node that stores the requested content object. Then, the node who has the requested content object creates a *Data* message containing the requested content object and sends it back to the client. A Data message is forwarded over the reverse path that the corresponding Interest has traveled. An important feature of CCN is in-network caching, i.e., CCN permits the intermediate routers to cache the Data messages that pass through them to reach the clients. Therefore, future repetitive Interest messages could be retrieved from closer caches.

In 2009, PARC developed an open-source software in C called CCNx 0.1 [1] to implement CCN [37]. In 2010, National Science Foundation (NSF) funded ten institutions, including PARC, to continue protocol design and software development of the CCN architecture, but the project was renamed to Named Data Networking (NDN) [4]. From 2010 to 2013, the NDN project used CCNx (versions 0.3 to 0.8) as its implementations. In 2013, PARC decided to continue developing CCNx as a closed-source software for commercial use. However, the NDN project team aimed to develop a framework for research with open-source software. Therefore, the NDN project team separated its software development activities and implemented: 1) NDN C++ library with eXperimental eXtensions (NDN-CXX) [56], and 2) NDN forwarding Daemon (NFD) [8]. NDN-CXX implements NDN primitives, i.e., name, Interest, Data, face, and signature. NFD implements the forwarder of NDN. Therefore, NDN-CXX and NFD form the open-source software of NDN.

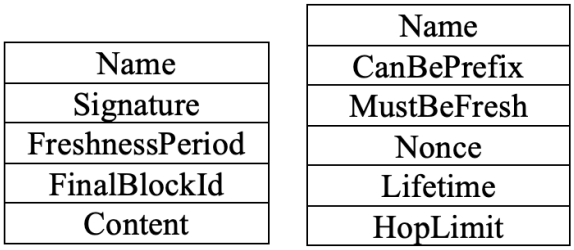
NDN and CCNx are designed with the same set of principles, e.g., 1) named-based routing, 2) in-network caching, and 3) content-oriented security [36]. Nevertheless, there are minor differences between CCNx and NDN [61]. For example, when CCNx routers want to check matching Data messages, they perform exact matching operations, while in NDN, when a client issues an Interest message, it might permit the Interest name to be a prefix or the exact name of the requested Data message. NDN uses random nonce values for loop detection, while CCNx does not provide loop detection capabilities. [36] describes a detailed description of the commonalities and differences between NDN and CCN.

To run simulations based on CCNx, users need to use CCNx with the NS-3 Direct Code Execution (DCE) project [6]. DCE is an NS-3 module, which aims to run network protocol implementations in the NS-3 simulator [5]. NDN project team provides ndnSIM [56] as an open-source simulator, which is based on NS-3. In this thesis, we focus on NDN [88] and we implement our protocols in ndnSIM [56].

2.2.6 Named Data Networking

Fig. 2.1a shows the Data message structure. A Data message requires a hierarchical *Name* for identification and a *Signature* for verifiability. However, a Data message might contain optional information, namely: 1) *FreshnessPeriod*, 2) *FinalBlockId*, 3) *Content*. Servers sign their Data messages using their secret keys. Therefore, clients can verify the Signature of a received Data message using the public key of the server that provides the received Data. If a content provider writes a FreshnessPeriod into a Data message, the Data message will become non-fresh after that period. A non-fresh Data is still a valid Data and FreshnessPeriod simply means that the server might have generated a newer Data after this time period. If a client receives a Data message that has the *FinalBlockId* enabled, the client considers the received Data message as the final segment of a data block. Finally, *Content* is an arbitrary sequence of bytes that the server might have written into the Data message as its content. FreshnessPeriod and FinalBlockId are not present by default, and the default value for Content is 1024 bytes. In this thesis, we consider the default settings for optional information.

Figs. 2.1b shows the Interest message components. It is mandatory for an Interest message to have a *Name* of a requested Data. However, an Interest message might contain optional information, namely: 1) *canBePrefix*, 2) *MustBeFresh*, 3) *Nonce*, 4) *Lifetime*, 5) *HopLimit*. If *canBePrefix* is disabled, Interest name has to fully match the Data name. However, if *canBePrefix* is enabled, the Interest name could partly or fully match the Data message. If *MustBeFresh* is enabled in an Interest, the Interest could be satisfied only with fresh Data messages. The *Nonce* is a randomly generated 4-bytes value. The combination of Interest Name and the Nonce value uniquely identifies an Interest message and is used for detecting loops. The *Lifetime* indicates the time period that an Interest message is stored in the PIT after its reception. The *HopLimit* is a 1-byte value, which defines the maximum number of hops that the Interest can travel. When a node receives an Interest, the node decreases the HopLimit value by 1, and if the resulting HopLimit is zero, the node will satisfy the Interest from the CS if



(a) Data message (b) Interest message

Figure 2.1: NDN message structures

possible and drops the Interest, otherwise. CanBePrefix, MustBeFresh, and HopLimit are not used by default. Nonce carries a randomly generated 4-byte value by default, and the default value of Lifetime is 4 seconds. We use these default settings in our simulations.

Fig. 2.2 depicts NDN Data message processing. NDN data forwarding is receiver-driven, i.e., Data messages are only sent in response to the received Interest messages; a received Data message for which no Interest message is stored will be considered unsolicited and will be discarded. In NDN, when a solicited Data message passes through a router, the router caches it. The cached Data messages are used to serve future matching Interest messages. To deal with the limited sizes of CS's, it is possible to use different content replacement strategies, such as Least Recently Used (LRU), Least Frequently Used (LFU), and Random [88].

Fig. 2.3 illustrates Interest message processing. This figure shows that when a router receives an Interest message, it checks whether the requested Data message exists in the CS. If there is a CS hit, the Data message will be returned. Otherwise, the router checks whether the Interest message is stored in the PIT and has been received before. If the PIT stores the Interest, the router aggregates the Interest message. Otherwise, the router checks the FIB to see whether next hop face(s) are available for the Interest. If the FIB has the next hop face(s) information for the Interest, the router dispatches the Interest to the *forwarding strategy*. The decision whether to forward an Interest message over each next hop face specified by a FIB is made by the forwarding strategy. NDN provides Multicast, Best route, and Random forwarding strategies [88]. Multicast forwarding strategy forwards Interest messages over multiple next hop faces to support multi-path content retrieval. Best Rote forwarding strategy forwards the Interest messages over the face with the lowest routing cost. Routing

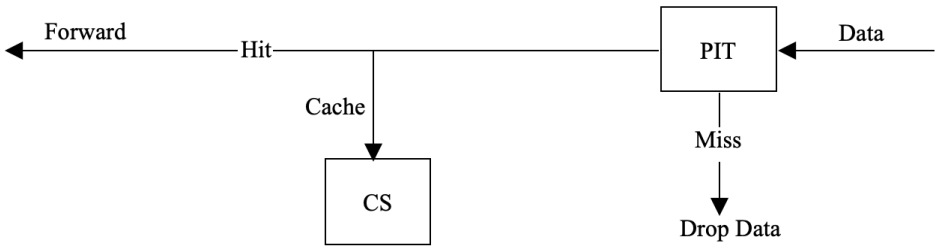


Figure 2.2: Data message processing

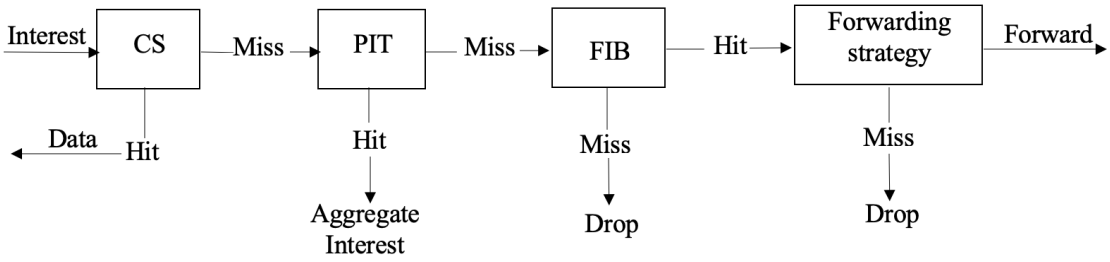


Figure 2.3: Interest message processing

cost for a face could be defined according to a metric, e.g., hop count to the requested content object. When the Random forwarding strategy is responsible for forwarding an Interest message, it randomly chooses one of the faces that are available for the Interest in the FIB and forwards the Interest message over the selected face.

NDN provides loop detection for Interest messages using random nonce values. Since Data messages travel over the reverse path of their corresponding Interest messages, Data messages will not be in loops. As Fig. 2.4 shows, each PIT entry maintains two data structures called *in-record* and *out-record*. In-records contain information regarding the faces over which the Interest message has been received as well as the last nonce value received over each face, and out-records contain information regarding the faces over which the Interest has been forwarded as well as the last nonce value forwarded over each face.

2.3 Routing

From among ICN architectures, DONA [41], PURSUIT [74], and NetInf [28] use name resolution for content discovery. Resolution-based content discovery maps the content requesters with content providers at rendezvous points [27, 63, 70, 79]. For this purpose, resolution-based content discovery solutions require complete informa-

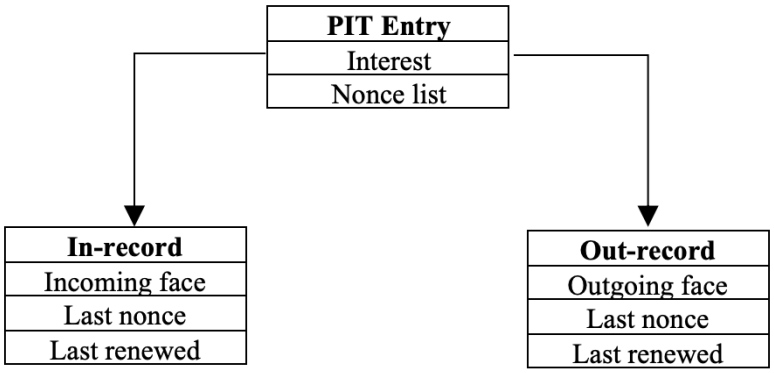


Figure 2.4: PIT entry structure

tion about the topology of the network as well as the distribution of content objects in the network. However, the maintenance of this information is not scalable [78]. Routing-based content discovery solutions mostly do not require full knowledge of the topology and the content distribution, and, therefore, routing-based content discovery solutions are more scalable [78]. CCN [37] and NDN [88] use routing-based solutions for content discovery [88].

In recent years, many routing protocols have been proposed for NDN [24, 35, 45, 49, 72, 77, 80, 86]. Wang et al. propose OSPFN [77], which is an extension to Open Shortest Path First (OSPF), as a routing protocol for NDN. OSPFN makes use of OSPF's Opaque (Link-State Advertisements) LSAs [15] for advertising name prefixes in the routing messages. OSPFN considers the best next hop for each name prefix. However, it can consider alternative next hops as well. OSPFN has the following shortcomings: 1) it requires IP addresses to identify routers, and 2) it is a single-path routing protocol. The work in [35] proposes Named-data Link State Routing (NLSR) as a link-state routing protocol for NDN, which uses LSA messages to exchange information about the available name prefixes as well as the topology of the network. In NLSR, each node runs Dijkstra's algorithm to find the shortest paths from each of the faces for any incoming Interest using full information about the topology and the content object name prefixes that exist in the network. NLSR proposes a hierarchical trust model to verify the authenticity of LSA messages in a single domain.

The works in [64, 68] use a Flooding-assisted Routing (FaR) strategy to find Data message delivery paths. FaR floods an Interest message when there is no routing information available for its name prefix (this is especially the case in the beginning of network operation). Therefore, clients and routers flood the Interest messages so

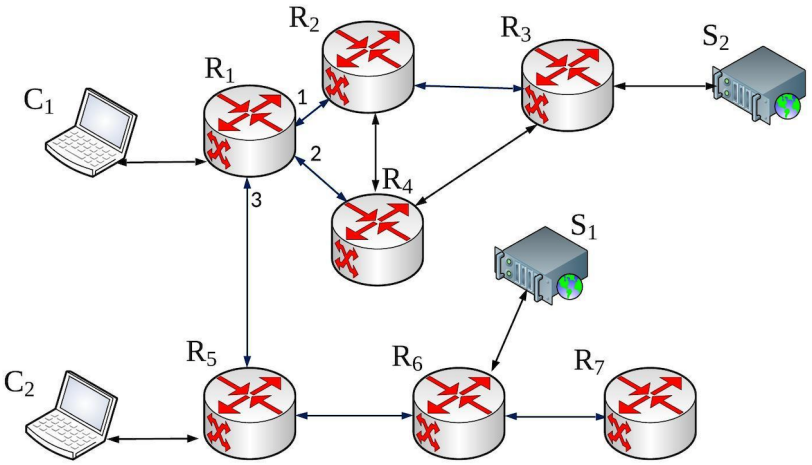


Figure 2.5: A topology for describing Flooding-assisted Routing

that they reach origin servers. When origin servers return Data messages, clients and routers receive them and populate the FIBs for the name prefixes of the received Data messages. Let us explain FaR with the help of Fig. 2.5. In Fig. 2.5, we assume that client C_1 issues an Interest message I_1 to demand a content object with name N_1 provided by server S_1. However, client C_1 does not have any information about the provider of content object N_1. Thus, client C_1 floods Interest message I_1 and waits for the Data message with name N_1. Since server S_1 provides the Data message with name N_1, client C_1 receives this Data message over the path $S_1 - R_6 - R_5 - R_1 - C_1$. When the Data message passes through routers R_6, R_5, and R_1, each of these routers populates the FIB for name N_1. For example, when router R_1 receives the Data message over face 3, router R_1 populates the FIB for name N_1 and considers face 3 as the only next hop face for name N_1 in the FIB.

The work in [80] proposes to use BFs for content advertisements from routers and to use an IP-based fall-back mechanism (possibly IP-based) to complement this scheme. Nevertheless, in CCN and NDN, temporary copies of content objects are cached en-route to the permanent copies, thus it is enough to only advertise permanent copies of content objects. In [39], it is assumed that initially Interest messages are flooded. However, when an Interest reaches a router that holds a copy of the demanded content object, the router piggybacks a BF containing the file names cached locally to the Data message and forwards the Data message back towards the requester node. The nodes that receive this Data message retrieve the BF from the Data message and use the BF for routing future incoming Interest messages.

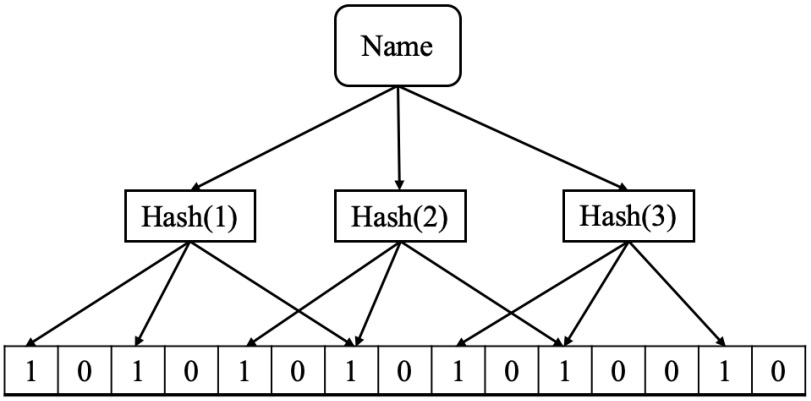
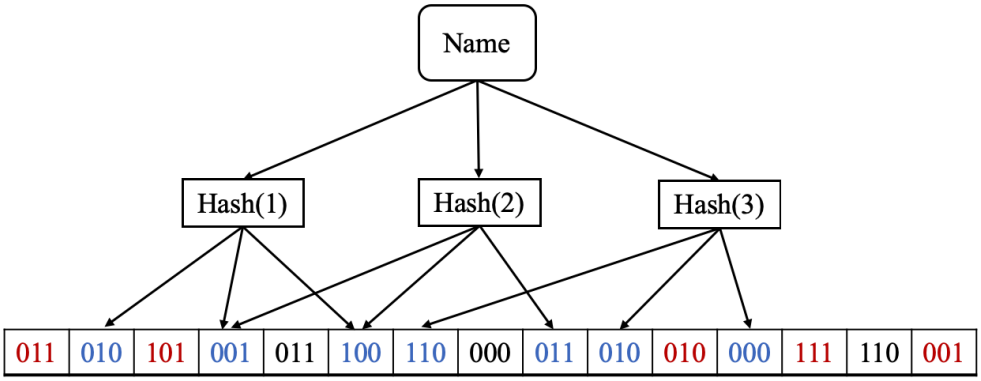


Figure 2.6: A BF with three hash functions

SCAN [45] is a BF-based routing protocol for ICN, but it is not a fully content-oriented routing protocol, because it uses IP routing as a fall-back approach. SCAN uses BFs to represent the content objects' names passing through each interface. The main drawback of this approach is that the number of elements inserted to the BF associated with the interface increases with the number of content objects passing through each interface. This will result in all the BF bits to be set to 1. Thus, the BF will not represent the elements that have passed over the interface correctly. To cope with this problem, COntent-oriented intra-domain Bloom filter-based Routing Algorithm (COBRA) [72] proposed Stable Bloom Filters (SBFs) [29] to represent the content objects' names passing through each interface. SBFs need much more storage overhead compared to standard BFs. However, they maintain only the names of the content objects that recently passed through each interface and discard the names of older content objects randomly.

To the best of our knowledge, BFR and COBRA are the only BF-based routing approaches proposed for NDN that are fully content-oriented and do not need any fall-back routing schemes as a complementary component of the routing process. Thus, in Chapter 3 we will compare BFR and COBRA. In the following, we describe BFs, SBFs, and COBRA operations.

Bloom Filter: BF is a space-efficient data structure to represent sets compactly and to support membership queries. When one represents a set with a BF, false positive probability impacts the performance of the BF, i.e., the probability that an element that is not in the set is wrongly reported by a BF as being in the set. In [17], the false positive probability is expressed as a function of the length of bit table m, the length of



(a) Before the insertion operation

(b) After the insertion operation

Figure 2.7: An insertion operation for an SBF with parameter set $\{m = 15, k = 3, d = 3, p = 5\}$

the original set represented by BF n, and the number of hash functions k. According to [17], when one wants to insert n elements in a BF and can afford a false positive probability p, the required size for the bit table m and the number of hash functions k are respectively given as:

$$\begin{cases} m = -\frac{nln(p)}{(ln2)^2} \\ k = \frac{m}{n} ln2 \end{cases} \tag{2.1}$$

Fig. 2.6 shows a BF with a 15-bits table and three hash functions. The BF's bit table is initialized by zero. The insertion operation of an NDN name consists of giving the NDN name as an input to the three hash functions to receive three positions in the bit table and set all the bits at those positions to 1.

Stable Bloom Filter: In contrast to BF that is a table of bits, SBF is a table of counters. Assume that an SBF consists of n counters $SBF[1], .., SBF[n]$ and the length of each counter is d bits. Thus, the minimum and the maximum values for each counter are 0 and $2^d - 1$, respectively. SBF is a variant of BF, which also uses hash functions for insertion and query operations. Like in standard BF, SBF's table is also initialized by zero. However, the insertion operation for SBF differs from the insertion process of BF.

When one wants to insert an element into an SBF, it gives the element to the k hash functions and the k counters indexed by the outputs of the k hash functions are set

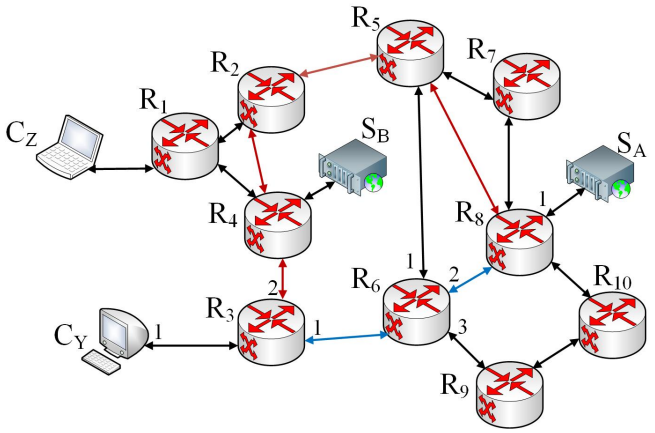


Figure 2.8: A topology for describing BFR and COBRA

to their maximum values $(2^d - 1)$. Then, p counters are randomly selected and their values are decremented. Fig. 2.7 illustrates the insertion operation for an NDN name into an SBF with parameter set $\{m = 15, k = 3, d = 3, p = 5\}$. This figure also shows the states of the SBF before (i.e., Fig. 2.7(a)) and after (i.e., Fig. 2.7(b)) the insertion operation, respectively. This insertion mechanism aims at keeping the elements that have been recently inserted into the SBF and removing elements that were previously inserted in the SBF randomly.

COBRA Operation: COBRA equips nodes with SBFs. Each node maintains as many SBFs as it has interfaces. When a content segment travels over a path towards a client, each node located on this path receives the content segment over an interface and stores the name of the content object as well as all the name prefixes of it into the SBF associated with the interface. This leads to storing a trace of the retrieval path for the content object at all nodes located on the retrieval path of the content object. In the beginning of the network operation, all SBFs of all nodes are empty because no content objects have been retrieved yet. Therefore, nodes do not have route traces stored in SBFs. Thus, nodes end up in flooding all the Interests until the SBFs learn the route traces. This phase is called *learning phase*. Flooding Interests during the learning phase does not scale well with increasing size of the content universe. In Fig. 2.8, assume that at time instance t_1, client C_Y issues an Interest to demand a content object segment named $/unibe.ch/images/fileName1/0$, which is stored at server S_A. When the SBFs are empty, both client C_Y and router R_3, as well as the other routers have to flood the Interest until it reaches server S_A. Then, this server sends the demanded Data message backward to client C_Y. Since the Interest has been flooded, the corresponding Data message comes back over different paths (e.g.,

path $S_A - R_8 - R_6 - R_3 - C_Y$ (blue in Fig. 2.8), path $S_A - R_8 - R_5 - R_2 - R_4 - R_3 - C_Y$ (red n Fig. 2.8)). Therefore, for router R_8, name $/unibe.ch/images/fileName1$ and all its name prefixes are inserted into the SBF of interface 1. But for router R_6, the same name prefixes are inserted into the SBFs of interfaces 1, 2, and 3. This is because router R_8 receives the Data message only from server S_A, while router R_6 receives the Data message from routers R_8, R_5, and R_9. After storing the route traces for name $/unibe.ch/images/fileName1/0$ and its name prefixes, client C_Y and all the routers that stored these name prefixes in their SBFs, do not need to flood anymore the Interests that come for the subsequent segments of the same file name.

When SBFs are not empty for a name prefix, another phase of COBRA routing called *interface ranking* takes place. Let us explain this phase with the help of the topology in Fig. 2.8. Let us assume that at time instance t_2, when the route traces for name $/unibe.ch/images/fileName1$ are stored in SBFs, client C_Y issues Interest I_1 to demand the subsequent segment of the same file name, i.e., $/unibe.ch/images/fileName1/1$. Client C_Y checks the full name against the SBF of interface 1. The SBF of interface 1 does not contain the full name. Thus, client C_Y increases the routing cost (initialized by zero) and eliminates the last name component. Then, client C_Y checks the resulting name against the SBF of interface 1. Since the SBF of this interface contains the name, client C_Y assigns the current routing cost in the FIB to interface 1. Client C_Y does not have any more interfaces. Thus, it forwards the Interest over interface 1 towards router R_3.

When router R_3 receives the Interest, it checks the full name against the SBFs of all the interfaces except the incoming one. Since the SBFs of interfaces 1 and 2 do not contain the full name, router R_3 increases the routing cost (initialized by zero) and eliminates the last name component. Then, router R_3 checks the resulting name against the SBFs of interfaces 1 and 2. Both of these SBFs contain the name. Thus, interfaces 1 and 2 are ranked with the current routing cost. The same process continues at the other routers until one of the following conditions happens: 1) all the interfaces of the router are ranked; 2) the last component of the Interest is eliminated and still one or more interfaces are not ranked. In the latter case, the router assigns the maximum routing cost to those not yet ranked interfaces.

To deal with link failures, COBRA permits nodes to reset the SBF(s) associated with a failed link upon detecting a link failure, i.e., setting all the bits of the SBF(s) to zero. Following this reset strategy, the nodes that are directly connected to the failed

link know that no content object is reachable through the failed link. Thus, they avoid forwarding any Interest messages over the failed link. When a link recovery is detected, COBRA allows nodes to set the values of all the counters of the associated SBF(s) to the maximum value. This strategy encourages the nodes that are directly connected to the recovered link to forward all Interests through the recovered link. The forwarding through the recovered link is temporary. When new content object names are inserted into the SBFs associated with the newly recovered link, the route traces will be corrected in the SBFs.

If content migration takes place, the route traces stored in SBFs should be corrected, because the location of permanent copies of the migrated content objects has changed and the temporary cached copies might be evicted due to a caching policy, e.g, LRU. With COBRA, clients and routers are not explicitly informed about a content migration event. However, they consider Interest retransmissions as indications of wrong forwarding decisions made in the past. Thus, when a node observes an Interest retransmission event, it retransmits the Interest not only over the interface with the smallest routing cost, but also over the interface(s) with higher routing cost(s) to increase the probability that the retransmitted Interest reaches the right server.

COBRA floods all the Interest messages during the learning phase. When Data messages are received, COBRA updates the route traces in SBFs. If route traces for an Interest's name prefix is stored in SBFs, COBRA avoids flooding the Interest and routes the Interest according to the route trace information.

2.4 Network Coding-based Content Retrieval

In their seminal work on Network Coding (NC), Ahlswede et al. [12] showed that the broadcast capacity of a network can only be reached with NC. This still holds true if the nodes restrict themselves to linear encoding functions [47]. Besides throughput gains, NC provides a minimum-energy multicast with a polynomial-time solution [83], which is NP-complete for classical routing. Optimal centralized NC algorithms have been the subject of several publications [38, 40]. [91] proposed a distributed implementation of global optimization in centralized NC. [81] showed that by choosing encoding functions randomly [25, 34], NC becomes very suitable for wireless ad-hoc networks. However, these approaches need to have a priori knowledge of the topology to be applicable [52].

Linear Network Coding (LNC) [47] is proposed as a means of code design to achieve the max-flow bound on the information transmission rate for multicast in networks. Nodes implementing LNC may send messages containing linear P combinations of earlier received information, i.e., $y_j = \sum_i \alpha_i x_i$ where the packets $x_i \in \{0,1\}^N$ (N bits packets). All coefficients α_i are generally numbers in a finite field (generally the Galois field $GF(2^k)$). Upon receiving a network coded packet, a node can retrieve the linear equation contained in the packet and reconstruct a linear equation system to solve. We say that a packet received by a node and containing a linear combination of packets is innovative if it increases the rank of the set of received packets at this node. Random Linear Network Coding (RLNC) [33] suggests that nodes choose the coefficients α_i randomly and independently inside a large enough Galois field [38]. When a node receives m linear combinations of n packets, it can decode them provided that the set of combinations has a rank n. If nodes use RLNC and select the coefficients α_i randomly in a finite field, the above condition will occur with a high probability for $m = n$ [38]. This appealing property of RLNC makes it interesting for decentralized networks [25, 34].

Several works have studied NC-based content retrieval in ICN. The work in [65] proposes NetCodCCN as an NC-based content retrieval protocol for CCN and discusses that, when we have multiple servers, multiple clients, and multi-path communications, NC-based content retrieval increases the throughput and resiliency to packet losses. Nevertheless, [65] evaluates a scenario that clients are only interested in a single file and all the links have the same capacity of 12 Mbps. On the contrary, in this thesis, we assume that clients are interested in different files and different links have different capacities. Further, [65] considers single-session NC [30], i.e., routers can only combine packets that belong to the same content object. Nonetheless, multi-session NC [19] permits network nodes to combine packets that belong to different content objects into network coded packets. Differently from [65], in this thesis, we propose a protocol that leverages BF-based information that has been propagated for content discovery to design multi-session network codes for content retrieval.

Video streaming is an application of ICN. Dynamic Adaptive Streaming over HTTP (DASH) [69] is an efficient video delivery protocol. When DASH is used, servers have to encode the videos in different representations (i.e., bitrate and resolution). When clients request videos, they decide the representation of their requested videos according to their available network and display resources. Therefore, clients start and control video streaming using the DASH protocol. Similar to DASH, content

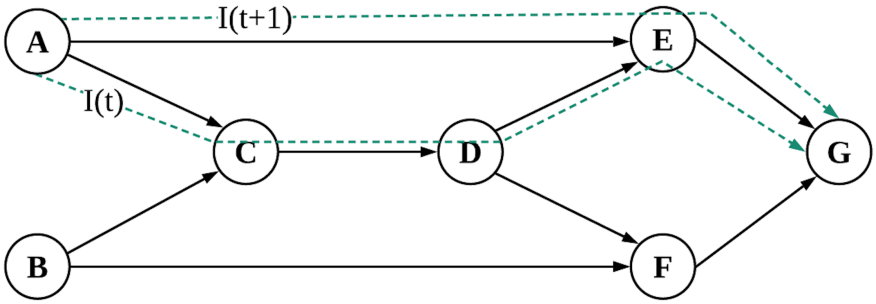


Figure 2.9: BF-based Interest message processing

retrieval is also client-driven in NDN. This interesting similarity between DASH and NDN motivated the work in [66] to propose a DASH-based video streaming protocol for NDN.

The work in [65] showed the NC gains in terms of throughput, delay, and resiliency, for multi-path communications scenarios in NDN. Therefore, the work in [66] implements NetCodNDN-DASH as a DASH-based video streaming protocol based on the NetCodCCN protocol [65]. The work in [66] shows that NetCodNDN-DASH achieves increased cache hit rate and reduced content traffic load on servers than the DASH protocol based on vanilla NDN, called NDN-DASH protocol. Further, if NetCodNDN-DASH is used, the number of clients that can receive the requested video with the highest available quality is significantly higher than of NDN-DASH [66].

Scalable Video Coding (SVC) [67] is another video streaming technique, which assumes that each video is encoded into different layers. In SVC, the lower layers have more importance than higher layers. Therefore, the first layer (base layer) has the highest importance, and users have to receive it before the higher layers. When a user receives the base layer of a video, the user can retrieve the lowest quality of the video. The more layers of a video a user receives, the higher is the quality of the video the user can retrieve.

Motivated by the NC gains in multi-path communication scenarios, the work in [20] proposes an architecture that makes use of NC techniques for SVC-based video retrieval [67] in NDN. [20] formulates a rate allocation problem to decide an optimal rate of Interest message transmission by clients to maximize the average quality of the scalable videos received by clients. If SVC is used, the users will have diverse requests in terms of video quality, and videos are divided into layers with inequal importance [67]. Therefore, to apply NC techniques to scalable videos, Prioritized Random Linear

Network Coding (PRLNC) was proposed to address the requirements of SVC in terms of video layers with inequal importance [43,71]. Since the NC method proposed in [20] is designed for SVC-based video delivery, it also uses PRLNC techniques [43, 71] to combine video packets.

Fig. 2.9 describes the Interest message processing proposed in [20]. In Fig. 2.9, we assume that client A issues two identical Interest messages $I(t)$ and $I(t+1)$ to request two linear combinations, with the same name prefix, at times t and $t+1$, respectively. As Fig. 2.9 shows, these Interest messages travel over the two green paths to reach server G. When vanilla NDN is used, if $I(t)$ reaches router E earlier than $I(t+1)$, then router E will assume that $I(t+1)$ is identical with $I(t)$, and, thus, router E will not forward $I(t+1)$ to server G. However, client A issued Interest messages $I(t)$ and $I(t+1)$ to receive two innovative network coded packets. To cope with this problem, the work in 2.9 suggests that both Interest messages $I(t)$ and $I(t+1)$ carry a BF containing the ID of client A. Then, when router E receives $I(t+1)$ after $I(t)$, router E will forward both $I(t)$ and $I(t+1)$, respectively, to server G. As a result, server G will return two innovative network coded packets to client A.

Although the work in [20] proposes to use BFs for Interest message processing, this work does not discuss the practical parameters and mechanisms related to BFs. The trade-off between BF size and the required bandwidth and storage resources, which we have clarified in this thesis, is not discussed in [20]. Further, the work in [20] permits intermediate routers to perform union operations on the BFs of Interest messages that have the same name prefix. Nevertheless, to permit BF union operations, it is highly important to design practical strategies that prevent BFs from overflowing after continuous union operations, which are not considered in [20]. In Chapter 4, we design practical BF aggregation strategies, which we also use in Chapter 5.

2.5 Service-Centric Networking

Service-Centric Networking (SCN) [21] is as an extension of ICN that permits clients to request services, which require service providers to perform computations on content objects. For example, a client might request a service provider to transcode a video as a service. SCN builds on top of CCN and the main principles of these architectures are the same. Specifically, in both architectures: 1) service request flow goes towards upstream, 2) service response flow returns over the reverse path of the

request flow to downstream. In SCN, service providers need to apply functions to users' or servers' data during processing time to generate the service requested by users. In the following, we briefly describe some of the related works on SCN.

Named Function Networking (NFN) [75, 76] proposes a service-centric architecture based on CCN [1]. NFN architecture is composed of an NFN layer that is responsible for forwarding decisions, and a service layer that runs the NFN engine (a.k.a. lambda engine) to perform computations on content objects. When a client wants to request computations on a content object, it sends the content object together with a function that has to be run on the content object. If an idle service provider receives a service request, it will perform the requested computations on the received content object and it will return the results. However, if a busy service provider receives a service request, it might forward the service request to other peers. Some of the SCN applications require to create a context and perform a series of operations in the created context. For example, communication parties might require to establish a session for exchanging symmetric encryption [18] keys before data communications and processes. However, NFN does not provide session support. To cope with this issue, the work in [32] proposes a session support mechanism for SCN. NFN does not have a hierarchical naming method. Thus, NFN names are hard to read especially when there are multiple function calls. To address this problem, [42] proposes the Named Function as a Service (NFaaS) architecture. NFaaS uses a hierarchical naming method rather than a lambda expression-based naming method. Further, NFaaS uses virtual machines, which could be implemented on any node, to implement functions. The developers of NFaaS claim that function migration is possible according to users' requests, however, the developers do not discuss the complexities related to function migration.

Similar to NFN, SOPHIA [82] makes use of a layered architecture for communications. A network layer forwards service data messages using IP addresses, and a service layer provides sessions for service processing. Before service provision, SOPHIA requires session establishments. When sessions are established, the network layer handles IP-based communications through the established sessions. The major shortcoming of SOPHIA is that it depends on IP-based communications, which does not comply with SCN design principles.

Differently from SOPHIA, Layered architecture for Service-Centric Networking (L-SCN) [31] presents a layered routing architecture for SCN, which is fully content-oriented and does not leverage IP addresses for routing. L-SCN designs intra and

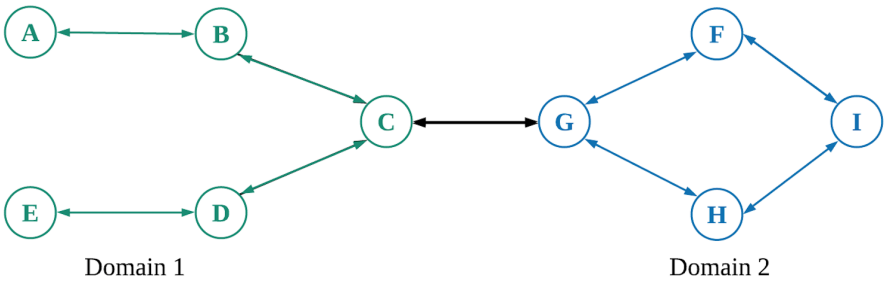


Figure 2.10: Domains and supernodes

Services	A BF containing available service names
Resources	Available processing and storage (e.g., CPU, RAM)

Figure 2.11: DIM structure

inter-domain communication mechanisms for service routing. For scalability reasons, L-SCN assumes that nodes are grouped into domains where each domain is managed by a special node called supernode. The supernode of each domain stores significant knowledge about the available services and resources of the domain. Further, the supernode of each domain is responsible for inter-domain communication with other domains. A supernode of a domain has to be connected with at least another supernode of another domain. Fig. 2.10 shows domains 1 and 2, where nodes *C* and *G* are their supernodes, respectively. If domain 1 requires information about the available services and resources in domain 2, supernode *C* has to request supernode *G* for this information.

For service routing, L-SCN suggests that in each domain, the supernode frequently requests the other nodes of the domain about the available services and resources. For example, in Fig. 2.10, supernode *C* sends an Interest Information Message (IIM) to ask the nodes of domain 1 (i.e., A, B, D, and E) about their available services and resources. When the nodes of domain 1 receive this IIM message, they reply with Data Information Messages (DIM) that contain the information about their available services and resources. As Fig. 2.11 shows, a DIM message stores a BF containing the names of the available services and also the information about the available resources.

Although L-SCN proposes to use BFs for service discovery, it does not discuss the practical details that are related to using BFs, e.g., BF size and its required bandwidth overhead, or false positive probability and its impact on routing. Further, L-SCN does not propose any method for grouping nodes into domains and for selecting

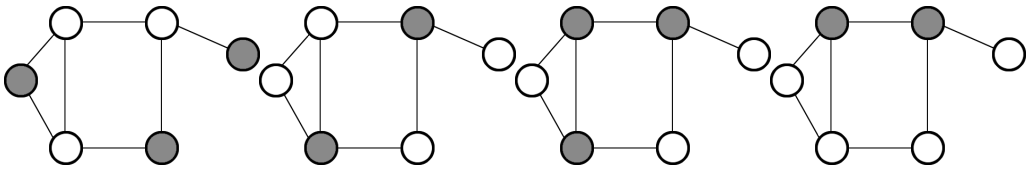


Figure 2.12: Different Dominating Sets. Dominator nodes are grey, Dominated nodes are white.

and connecting supernodes. In Chapter 6, we describe the practical parameters of the BFs and their impact on the performance. Further, we leverage dominating sets and connected dominating sets [44] to propose a solution for grouping nodes into domains, supernode selection, and supernode connection. In the following, we describe dominating sets and connected dominating sets.

2.6 Dominating Sets

For a graph $G =< V, E >$, where V is the set of vertices and E is the set of edges, a set $D \subseteq V$ is called a *Dominating Set (DS)* provided that each vertex of V is either an element of set D or it is directly connected to an element of set D. The elements of a DS are called dominators and the direct neighbors of dominators are called dominated nodes. The DS with the minimum possible cardinality is called Minimum DS (MDS). Given a DS $C \subseteq D$, if any vertex $v_i \in C$ can reach any other vertex $v_j \in C$ using a path that does not leave set C, the latter set is called *Connected Dominating Set (CDS)*. A CDS with the minimum possible number of elements is called Minimum CDS (MCDS). In Fig. 2.12, we illustrate examples of DS, MDS, CDS, and MCDS over a graph. The determination of MDS and MCDS are NP-hard problems [23].

DS and CDS find applications in several networking problems, e.g., creating a virtual backbone for routing in ad-hoc networks [13, 23, 46, 84, 92]. As finding an MCDS is NP-hard, the work in [23] proposed to approximate MCDSs, by finding dominating sets slightly larger than a dominating set with the fewest possible number of nodes. This set is used as virtual backbones for wireless ad-hoc networks in a unit-disk graph to alleviate broadcast storms. The authors of [23] present a distributed algorithm to approximate MCDS, which first finds the maximal independent set and then uses a Steiner tree to connect the vertices in this set. The work in [13] proposes another distributed algorithm that does not depend on spanning trees. This algorithm main-

tains the same approximation ratio after topology changes, but needs a leader node to operate. If a leader node is not given, leader election should take place which adds time and message complexity to the algorithm. In [92], a fully localized and distributed algorithm called r-CDS is proposed, which does neither require to build a tree nor to select a leader. The work in [46] examines distributed algorithms proposed for MCDS approximations and presents distributed construction of an approximate MCDS, for unit disk graphs.

The work in [84] uses DSs to provide a solution for collaborative caching in ICN. It leverages DSs for efficient collaborative caching to reduce caching redundancy in the network. It combines routing and caching strategies with a CDS. In [84], the most popular content objects are cached at the core routers. The benefit of having a CDS for the core routers is much simpler routing, as every core router is directly connected to at least another core router, and, therefore, routing can be done solely through core routers. To the best of our knowledge, the work in [84] is the only that makes use of DSs to provide a solution for ICN-based networks, but it uses a centralized clustering algorithm and requires the network topology as well as the number of neighbors for each node to be known a priori. The main problem with the work in [84] is that even if the CDS construction methodology can operate easily for small topologies, its complexity increases fast with the size of the network topology [53].

In [13, 23, 46, 92], the presented solutions are for wireless ad-hoc networks, modeled as unit disk graphs. Differently from these works, in Chapter 6, we focus on using DS and CDS for intra-domain and inter-domain routing and propose a fully distributed algorithms to construct these sets for any arbitrary network topology.

2.7 Conclusions

In this Chapter, we described the concepts that we use in the remainder of the thesis. Section 2.2 introduced different ICN architectures. We mentioned that we focus on NDN [88] as the most prominent ICN architecture and we use ndnSIM [56], which is the open source NDN simulator, for implementing our protocols. Next, in Section 2.3, we described the related works on routing in NDN and also introduced BFs that are space-efficient data structures to represent sets compactly. In the following Chapters, we make use of BFs for compressing the routing information in NDN and SCN. Then, Section 2.4 described network coding-based content retrieval and the

previous works on network coding-based protocols for ICN. In Chapter 5, we propose a novel NC-based protocol for reducing content retrieval delay. After, Section 2.5 introduced SCN and summarized its related works. Finally, Section 2.6 introduced DS and CDS concepts. In Chapter 6, we focus on SCN and we make use of the DS and CDS concepts for dividing network nodes into different domains, which is required before implementing intra-domain and inter-domain routing.

3

Push-based Bloom Filter-based Routing

3.1 Introduction

In this Chapter, we attempt to answer RQ 1 posed in the Introduction section that is related to the design of efficient protocols for routing Interest messages in NDN. To route Interest messages, routing protocols require to populate FIBs. Therefore, the development of strategies that optimally populate FIBs is vital for NDN [50]. The Interest flooding method is inefficient as it wastes significant bandwidth resources. Differently from flooding, shortest path routing solutions forward each Interest only over the shortest path to the origin server of the demanded content object. These routing solutions require full knowledge of the topology as well as the location of origin servers for all the existing name prefixes in the network that entails a large overhead.

To avoid wasting network resources through Interest flooding, an alternative approach is to permit origin servers advertising their content offers frequently, i.e., whenever new content objects are available in repositories. Therefore, origin servers could use BFs to compactly represent their content offers. This leads to smaller overhead needed

for the propagation of content advertisements. Due to these appealing features of BF-based content advertisement, in this Chapter we propose BFR, a routing protocol that uses BFs for content advertisements from origin servers for FIB population.

In NDN, temporary copies of a content object might be cached en-route to the nodes that provide the permanent copies of the content object. This possibility of in-network caching enables clients to retrieve content objects from the caches that are closer to them rather than from possibly distant servers. In our scheme, only origin servers perform BF-based content advertisements. Nevertheless, nodes receive the content advertisements of an origin server over all the paths en-route from the origin server and populate their FIBs accordingly. Further, we adopt the multicast forwarding strategy for forwarding Interests. Therefore, BFR forwards each Interest in parallel through all the paths towards the origin server of its demanded content object. The Interest could be satisfied with the caches before reaching the origin server. Hence, it is unnecessary for routers to explicitly advertise their cached content objects, like the scheme proposed in [39], and incur higher advertisement overhead.

Push-based BFR is topology oblivious. Hence, it does not need to propagate and store information about the topology that entails overhead. Further, push-based BFR uses BFs to reduce storage and signaling overhead of content advertisements. Finally, it does not adopt any IP-based routing protocol as a primary or fallback mechanism. This makes push-based BFR fully content-oriented and removes any dependencies on IP-based communication models.

The remainder of this Chapter is organized as follows. Section 3.2 describes the proposed push-based BFR method. Then, Section 3.3 discusses the impact of false positive errors on push-based BFR operation, robustness to topology changes, and handling of content migration. Afterwards, we present in Section 3.4 a simulation-based comparative analysis of the proposed push-based BFR against flooding, shortest path, and COBRA [72] routing protocols to illustrate push-based BFR advantages in practice. Finally, Section 3.5 concludes the Chapter.

3.2 Push-based Bloom Filter-based Routing

In push-based BFR, origin servers represent and advertise their content objects using BFs. In summary, push-based BFR consists of three phases: a) Representation of content objects using BFs, b) BF-based content advertisement, and c) FIB population

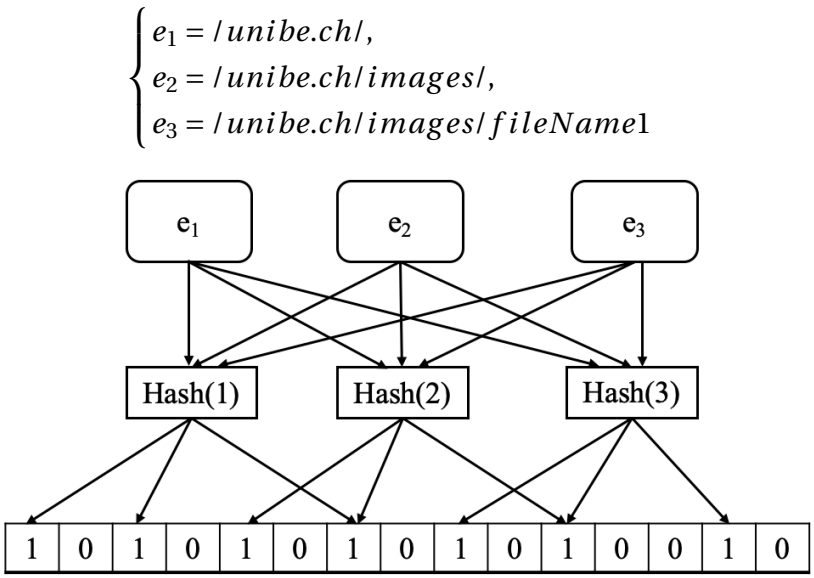


Figure 3.1: An example for content advertisement BF and related hash functions

and content retrieval. In the following, we describe each phase in detail.

3.2.1 Representation of Content Objects Using BFs

In push-based BFR, when an origin server has content objects to offer, it generates an empty BF for which all the bits of the bit array are set to zero. Then, the origin server maps the names of its content objects into the generated BF. An example of inserting three URLs into a BF with a parameter set $\{m = 15, n = 3, k = 3\}$ is presented in Fig. 3.1 (we defined m, n, and k in Eq. (2.1)). As Fig. 3.1 shows, the insertion process consists of feeding each URL to the three hash functions to get three positions in the bit array and set all the bits at these positions to 1.

In push-based BFR, each origin server maps the names of its content objects as well as their name prefixes in its BF. For example, as Fig. 3.1 shows, the full name (e.g., $/unibe.ch/images/fileName1$) as well as the name prefixes (e.g.,$/unibe.ch/$ and $/unibe.ch/images/$) are inserted into the BF. In Section 3.2.3, we discuss the reasons behind inserting name prefixes into BFs in detail.

To show the savings resulting from using BFs for representing a set of content objects, we provide an example. Consider that an origin server stores 200 content objects, which are each divided into a number of segments. To represent the content objects,

the server creates a BF by setting $n = 200$, and targets a false positive error probability of 2% (approximately four names per BF). Thus, the server needs a BF of size $m = 1628.47$ bits and $k = 5.64$ hash functions. Aligning the bit table size to byte order and rounding these values, the server requires 203.5 bytes, i.e., approximately one byte per named content object. For larger BFs, i.e., larger values of m and n, and the same false positive probability, the required space for inserting each URL into the BF stays constant, i.e., one byte. In NDN, names are URLs. To evaluate our routing approach, we consider a realistic URL catalogue [26] with the average URL size equal to 42.45 bytes. For this setting, a server needs 8490 bytes to advertise a list of 200 URLs without BF, while it needs only 2.4% of this size, i.e., 203.5 bytes, in case it uses BF. Therefore, the use of BFs results in high compression for representing a set of content objects.

3.2.2 BF-based Content Advertisement

When an origin server creates a BF that contains the names of its content objects, it requires to propagate this BF to advertise its content objects. We introduce a new type of Interest packet called Content Advertisement Interest (CAI) that carries content advertisement BFs. Hence, push-based BFR propagates CAI messages carrying the content advertisement BFs. The NDN Interest forwarding pipeline detects and discards duplicate CAI messages and ensures loop freedom for these messages. It is important to note that the only purpose for the propagation of CAI messages is content advertisement and no Data packet is sent as a response to CAI messages. Fig. 3.2 illustrates the structure of a CAI message, which is identified by the name prefix $/ContentAdvertisement$. To distinguish the CAI messages issued by different origin servers, we allow each origin server to append its unique ID as the second name component to the name of the CAI messages that it issues. In the forthcoming, we describe the reasons behind this choice in detail. As Fig. 3.2 shows, each CAI message similar to Interest messages exploits a random *nonce* to ensure loop freedom. The nodes that receive CAI messages store them in their PITs. CAI messages should expire like other packet types stored in nodes' PITs. Since no Data is coming back in response to the CAI messages, they stay in PITs until their timeout. Hence, it is necessary to add to the CAI message a *lifetime* field, which indicates when it expires. To this aim, we reuse the *Interest lifetime* field to indicate the lifetime of CAI messages. Origin servers refresh the CAI messages to keep nodes informed about their content offers. Further, the content advertisement applications do not re-express CAI messages. We should emphasize that this work aims at proposing a BF-based content advertisement

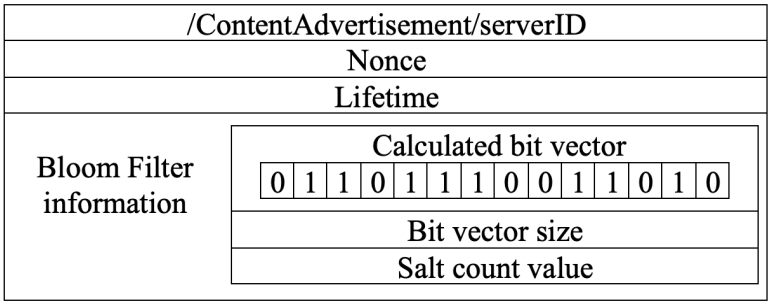


Figure 3.2: CAI message

strategy fully compatible with the original NDN and not to present an NDN variation. The last components for a CAI message are the needed information to retrieve the content advertisement BF consists of the calculated bit array, the size of the bit array, and a salt count value that is needed to retrieve the same content advertisement BF at the nodes that receive the CAI message. Here, we assume that all the origin servers generate their hash functions with a universal random seed and operate with the same set of hash functions.

To permit nodes to propagate CAI messages, we add a FIB entry for name prefix $/ContentAdvertisement$ in the FIBs of all the nodes and add all the faces as next hops for this name prefix at each node. Further, we adopt the multicast strategy for forwarding the $/ContentAdvertisement$ name prefix. Therefore, when an origin server issues a CAI message, this message is forwarded to all the nodes that are located in one hop distance and those nodes forward it over all the faces except the incoming one. Each node that receives the CAI message broadcasts it, while the Interest forwarding pipeline of the NDN Forwarding Daemon (NFD) ensures loop freedom and discards duplicate CAI messages. Therefore, all the nodes will eventually receive the CAI message.

The nodes that receive CAI messages record in their PITs the faces over which they receive each CAI message. In Chapter 2, we explained that, the faces over which an Interest is received are stored in the in-records of the related PIT entry. Therefore, to record the faces over which a CAI message is received, we use in-records.

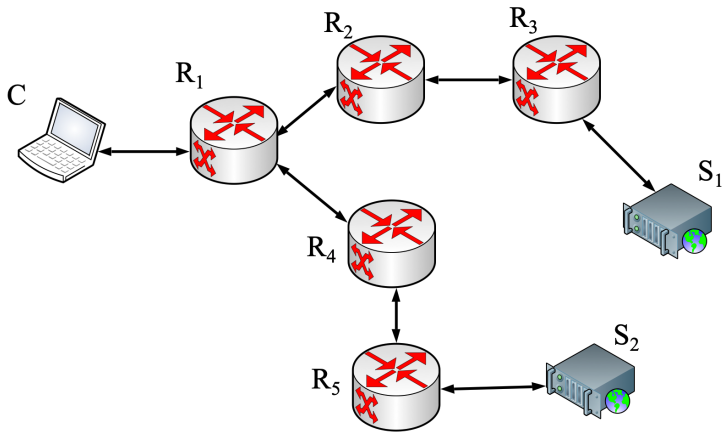
All the CAI messages share the same name prefix, i.e., $/ContentAdvertisement$. Nevertheless, we let origin servers append their $uniqueIDs$ as the second name component to the name of the CAI message. For example, in Fig. 3.3a server S_1 generates a CAI message with name $/ContentAdvertisement/S_1$ and server S_2 a CAI message

with name prefix $/ContentAdvertisement/S_2$. In general, servers could append any kind of unique ID (*e.g.*, their MAC addresses) as the second name component to ensure name uniqueness. Let us provide an example to explain the reason behind appending $serverIDs$ as the second name component for CAI messages. In Fig. 3.3a, assume servers S_1 and S_2 do not append their unique IDs as the second name component to CAI messages. In such a case, server S_1 sends a CAI message with name $/ContentAdvertisement$, and router R_1 receives it. If at a later time instant, server S_2 sends a CAI message with the same name, which is received also by router R_1, the NFD Interest forwarding pipeline will consider the second CAI message received by router R_1 as redundant because both messages have the same name. This will lead router R_1 to only record the incoming face of the CAI message issued by server S_2 in the PIT entry for name prefix $/ContentAdvertisement$ and to discard it. This approach makes router R_1 to discard the content advertisement BF of server S_2. Hence, router R_1 will be unaware of the content offers from server S_2. To avoid this problem, in push-based BFR origin servers append their $uniqueIDs$ to the name of CAI messages.

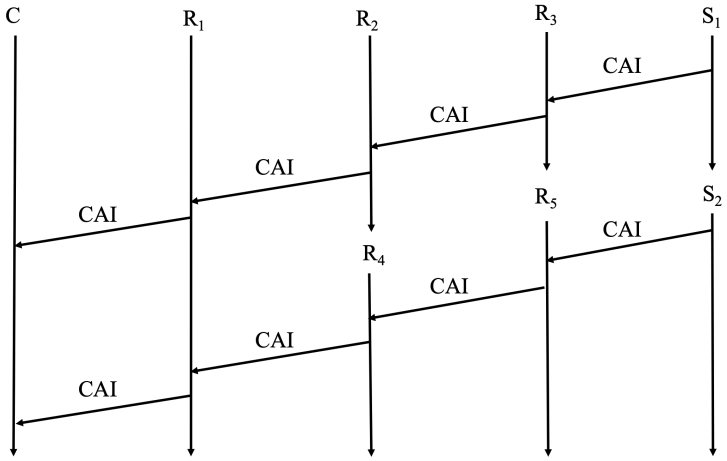
Fig. 3.3b illustrates the content advertisement process. We assume that in Fig. 3.3a server S_1 advertises its content objects by sending a CAI message to router R_3. Router R_3 receives and stores this message in its PIT, and forwards it to router R_2. Router R_2 also stores the CAI message in its PIT and forwards it to router R_1. This is done until all the nodes obtain the CAI message. At the end of this process, all the nodes receive the CAI message issued by server S_1. Fig. 3.3b also shows the content advertisement process of server S_2. In general, CAI messages could flood the network, or could be sent using random walk. Although the random walk strategy incurs less bandwidth and storage overhead, we did not follow this strategy because not all the nodes will be aware of the content objects offered by all the origin servers.

3.2.3 FIB Population and Content Retrieval

Push-based BFR combines FIB population and content retrieval processes. To describe the FIB population process, assume that server S_2 in Fig. 3.3a also advertises its content offers. After the completion of the content advertisement propagation from servers S_1 and S_2 at time instant t_2, all the nodes store the CAI messages $/ContentAdvertisement/S_1$ and $/ContentAdvertisement/S_2$ in their PITs. The PIT of client C is presented in Table 3.1. This Table shows the CAI messages in the



(a) A topology to describe push-based BFR



(b) CAI transmissions

Figure 3.3: Content advertisement

upper rows of the PIT to indicate that the CAI messages are distributed proactively. In push-based BFR, nodes use the received CAI messages for FIB population. When a client issues an Interest to retrieve some Data, FIB population occurs hop by hop at all the nodes that are placed on paths en-route to the origin server of the demanded Data.

Let us describe FIB population, by considering the topology presented in Fig. 3.3a and assuming that at time t_3, client C issues the first transmission of the Interest $/unibe.ch/images/fileName1/01$ to retrieve the first segment of content object $/unibe.ch/images/fileName1$ that is offered by server S_1. To populate its FIB, client C eliminates the sequence number from the name of the issued Interest and checks whether the BFs of the stored CAI messages contain name prefix

Table 3.1: PIT table of C

$/ContentAdvertisement/S_1$
$/ContentAdvertisement/S_2$
$/unibe.ch/images/fileName1/01$

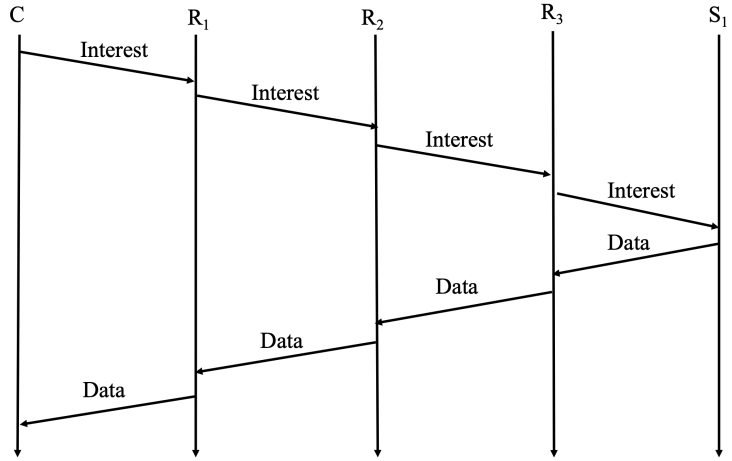


Figure 3.4: Content retrieval

$/unibe.ch/images/fileName1$.

In this case, the demanded content object is produced by server S_1, so the BF stored in $/ContentAdvertisement/S_1$ verifies that it contains the name prefix $/unibe.ch/images/fileName1$. Now, client C can add the face(s) over which it has received content advertisement $/ContentAdvertisement/S_1$ as the next hop faces for name prefix $/unibe.ch/images/fileName1$ into the FIB. Therefore, if no FIB entry exists for this name prefix, client C creates a FIB entry for this name prefix and adds the face(s) stored in the in-records for the CAI message $/ContentAdvertisement/S_1$ as the next hop face(s) for the FIB entry. After client C has populated its FIB for name prefix $/unibe.ch/images/fileName1$, it forwards the Interest for this name prefix to router R_1. Router R_1 runs the same process as client C and checks the Interest name without the sequence number, i.e., $/unibe.ch/images/fileName1$ in the BF of CAI messages stored in its PIT. Router R_1 forwards the Interest to router R_2, then router R_2 forwards the Interest to router R_3, and, finally, router R_3 forwards the Interest to server S_1 and the demanded content object is retrieved. Fig. 3.4 shows the described Interest and Data message flows. The first transmission of Interest $/unibe.ch/images/fileName1/01$ in the network should reach server S_1 to retrieve

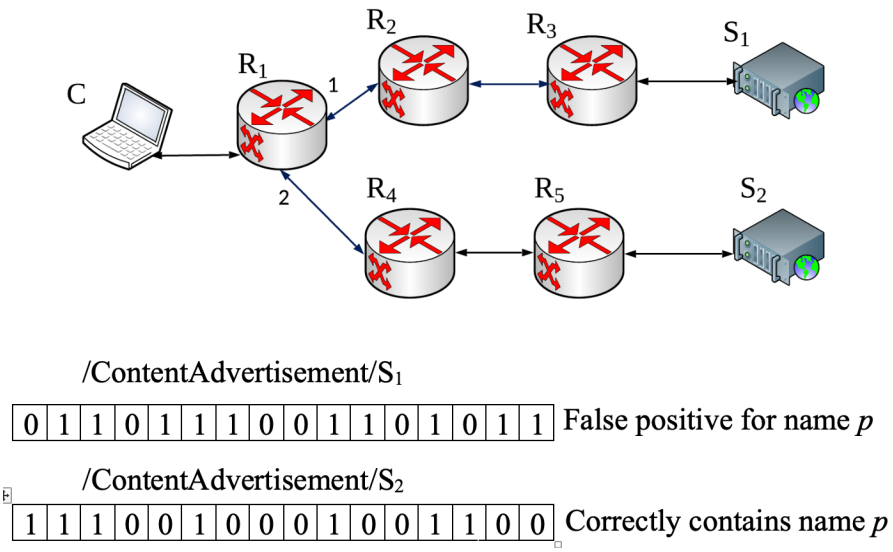


Figure 3.5: False positive error in content advertisement BFs

the demanded content object. The next transmissions of this Interest may retrieve the content object from closer caches at routers situated en-route the upstream path towards server S_1. We select the *multicast* forwarding strategy that forwards the received Interests over all the next hops specified in the FIB for their names and design push-based BFR to work with this strategy to benefit from the existence of multiple paths between the clients and the content servers. This approach is very efficient in case of topology changes, i.e., unexpected link failures or link recoveries, or when the shortest paths are congested, thus not able to return Data packets fast enough.

3.3 Discussion

In the following, we discuss the impact of false positive errors on push-based BFR operation, robustness to topology changes, and handling of content migration.

3.3.1 Impact of False Positive Errors on Push-based BFR Operation

When we use BFs, false negative errors cannot happen. However, false positive errors are possible and might affect the performance of the system. We assume that all the content advertisement BFs operate with the false positive probability of P_{fpp} and use Fig. 3.5 to discuss the impact of false positive errors on push-

based BFR operation. In Fig. 3.5, client C issues Interest I_p for name prefix p = /$unibe.ch$/$images$/$fileName1$, while server S_2 possesses the content objects for this name prefix. Client C forwards Interest I_p to router R_1. Router R_1 receives the Interest for name prefix p from client C. At this router, the content advertisement BF of CAI message /$ContentAdvertisement$/S_2 correctly verifies that it contains name prefix p and, therefore, router R_1 forwards the Interest for this name prefix towards server S_2. However, at the same router, the content advertisement BF of CAI message /$ContentAdvertisement$/S_1 might give a false positive report for name prefix p. Therefore, router R_1 might forward the Interest for name prefix p towards server S_1 as well. Since false negative errors are impossible with BFs, routers R_4 and R_5 continue forwarding Interest I_p towards Server S_2. Hence, Interest I_p will be eventually satisfied because server S_2 stores the requested content object. If false positive errors happen at both R_2 and R_3, these routers wrongly forward Interest I_p up to server S_1, which does not provide the requested content object. In Section 3.4.9, we present results to show the impact of false positive errors on push-based BFR routing in practice.

3.3.2 Robustness to Topology Changes

To combat link failures, routing protocols should be resilient to link failures and should adapt to link recoveries. When a link failure is detected, the nodes connected to the failed link should prevent Interests from passing through this link until it recovers. This is done in push-based BFR by taking the following actions, when a node detects a link failure: a) the node removes the face associated with the failed link from all the in-records of all the CAI messages that exist in the PIT, and b) the node removes the face associated with the failed link from all the FIB entries.

When detecting a recovered link, the nodes connected to this link force all the Interests to pass through it as it is a newly allocated network resource. In push-based BFR, the nodes connected to a recovered link perform the following actions: a) they add the face associated to the recovered link to all the in-records of all the CAI messages that exist in the PIT, and b) they add the face associated to the recovered link as a next hop face in all the FIB entries. The Interests pass through a recovered link for a short time because by receiving fresh CAI messages, all the routes will be automatically updated.

3.3.3 Handling of Content Migration

Content migration, i.e., moving a number of content objects stored in the repository of a server to the repository of another server, may occur. When content migration happens, it is necessary to propagate new CAI messages and to immediately inform the network about the changes in the servers' repositories so that nodes remove the stale CAI messages stored in PITs. For this purpose, we present a strategy, which aims at removing stale CAI messages from PITs upon detecting a content migration event. Let us explain our strategy by considering again the topology illustrated in Fig. 3.3a. Assume that client C maintains CAI messages from servers S_1 and S_2 in the PIT. If server S_1 migrates content objects to server S_2, these servers immediately propagate new CAI messages to inform all the network nodes about this event. However, servers S_1 and S_2 should not only update the nodes with new CAI messages, but they should also signal them to discard the CAI messages received before. For this reason, we enable servers to do this by adding a new flag called *discardOldAdverts* to the new CAI messages. Therefore, servers S_1 and S_2 activate the *discardOldAdverts* flag for the new CAI messages and propagate them. When client C receives the new CAI messages in which the *discardOldAdverts* flag has been activated, it removes all the CAI messages received in the past, which have been issued by S_1 and S_2 from its PIT, and stores the new CAI messages.

When an origin server replicates content objects to cache servers, these cache servers also should advertise their content objects. If the content advertisement BF of a cache server is identical with a content advertisement BF of an origin server, the nodes that receive these identical BFs can aggregate them. Origin servers might add or remove content objects to/from their repositories. If an origin server adds content objects to the repository, it advertises the fresh content objects at the next content advertisement round. If an origin server removes content objects from the repository, the removed content objects will not be inserted in the content advertisement BF next time that the origin server advertises its content objects. In case an origin server receives an Interest for a content object that it has removed recently, the origin server returns a "No Data" NACK [87] to announce the removal of the demanded content object.

3.4 Performance Evaluation

We compare push-based BFR with three other routing approaches: 1) *flooding*, where an incoming Interest is forwarded to all the faces except the incoming one, 2) *shortest path*, where the Dijkstra algorithm is employed to calculate the shortest paths, in terms of least number of hops to the origin servers, and 3) *COBRA* [72]. In Chapter 2, we described COBRA [72] and we mentioned that push-based BFR and COBRA [72] are the only BF-based routing protocols proposed for NDN that are fully content-oriented, topology oblivious, and distributed. Push-based BFR routes Interests according to push-based content advertisements, whereas COBRA routes Interests according to path traces left from previously retrieved content objects. In this Chapter, we show that even if push-based BFR requires nodes to exchange BF-based routing information, it still incurs much less communication overhead than COBRA, because in COBRA the nodes flood with Interests the network during the learning phase. Further, we show that in COBRA nodes need significantly more memory for storing routing information than push-based BFR. This is problematic if nodes have limited memory space. We implemented push-based BFR, flooding, shortest path routing, and COBRA in the ndnSIM2.1 [56] environment.

3.4.1 Simulation Settings

To evaluate all the schemes, we use two topologies: 1) the GEANT network topology depicted in Fig. 3.6, and 2) a 10×10 grid topology depicted in Fig. 3.7. The GEANT network topology [2] interconnects Europe's National Research and Education Networks (NREN) and provides research network services across the continent. Since the GEANT topology is tree-like, we also use the grid topology to assess the performance of all routing protocols when the topology is more connected. In the GEANT and the grid topologies, we distribute the endpoints, i.e., clients and origin servers, randomly in each simulation. There are five origin servers, which we randomly place in the GEANT and the grid topologies for each simulation. As for the clients, we attach a variable number of nodes (between three to six nodes) to each randomly selected router. Our GEANT and grid topologies contain 56 and 25 clients, respectively. Thus, the considered GEANT topology has 101 nodes, and the grid topology has 131 nodes.

We use a dataset of URLs extracted from real traces of HTTP requests [26]. We consider a content universe, i.e., the set of produced files at origin servers, that includes $100,000$

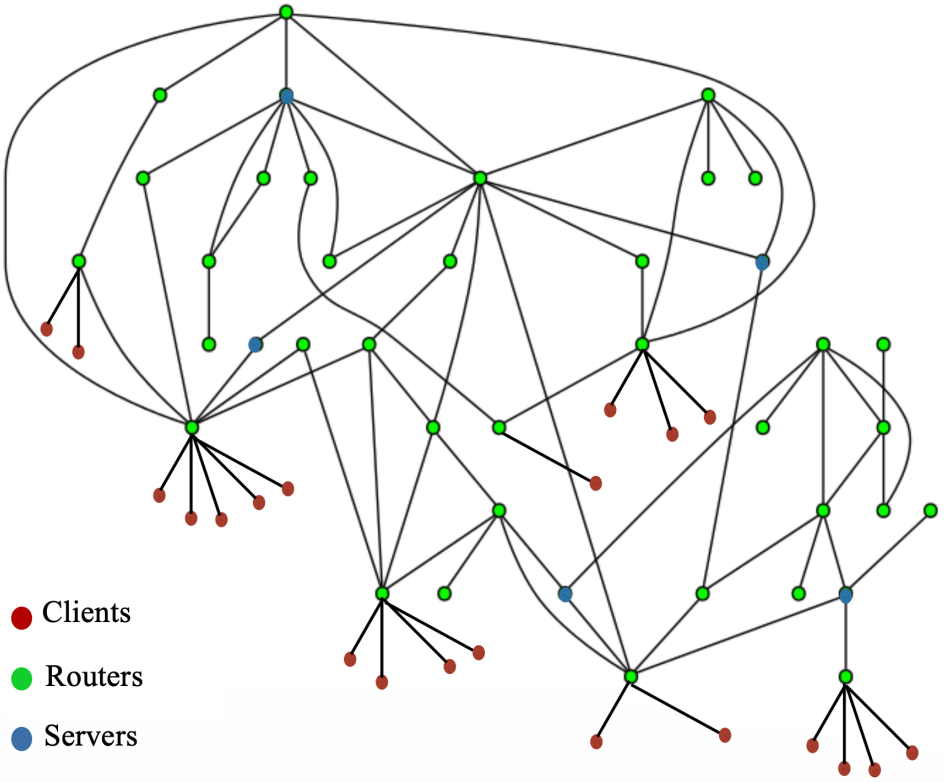


Figure 3.6: Geant topology and connected endpoints

file names in total. Each file has 100 segments. Thus, we generate 10^7 unique segments. We assume that the content popularity follows the Zipf-Mandelbrot law [58], which is shown in Eq. (3.1), where M denotes the cardinality of a content catalogue and α is the skewness of the popularity function (larger α values correspond to fewer popular content objects).

$$P(x = i) = \frac{1/i^{\alpha}}{\sum_{j=1}^{M} 1/j^{\alpha}} \tag{3.1}$$

For performance evaluation, we consider values of α in the $[0.8, 1.4]$ interval. All the results are averaged over ten simulations, each lasted for $100,000$ seconds. We report the average measured values over these simulations. The reported mean values have 95% confidence intervals. For BFs, we use the parameters $n = 1000$ and $p_{fpp} = 0.0638$, where n denotes the number of inserted elements in the BF and p_{fpp} denotes the false positive errors' probability. Therefore, the size of each advertised BF is roughly 716 bytes for advertising 1000 URLs. To create SBFs, we use parameter set $\{n = 10^6, p_{fpp} = 0.0638, d = 3\}$. Similar to BFs, n and p_{fpp} denote to the number

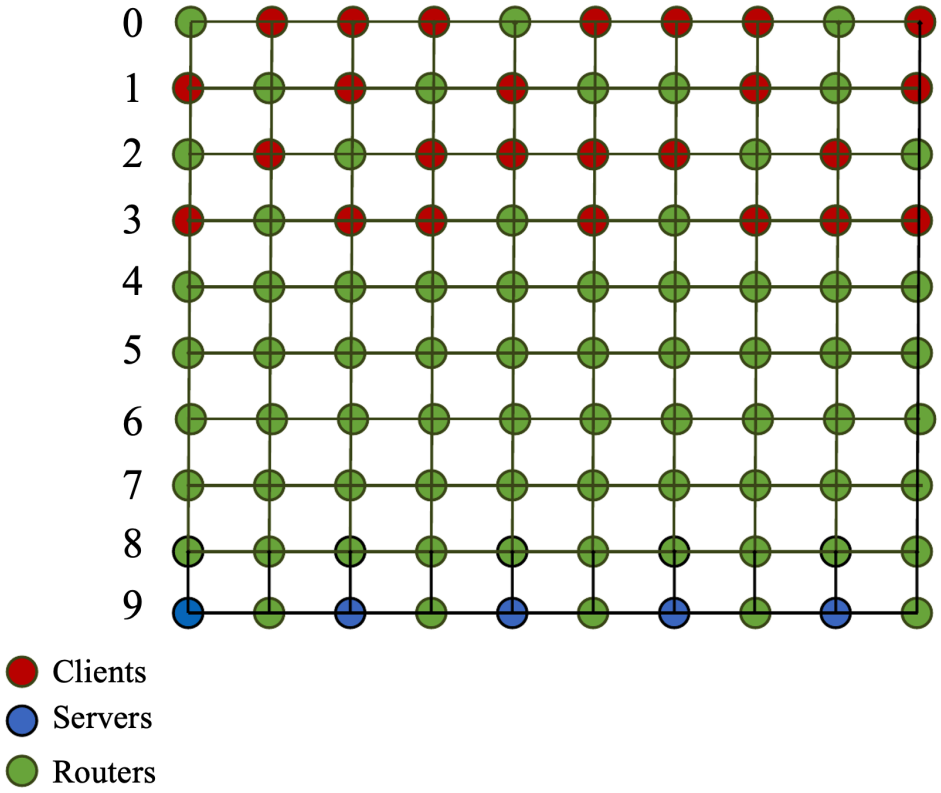


Figure 3.7: Grid topology and attached clients and servers

of inserted elements and false positive errors' probability, and d denotes the length of SBF counters. Therefore, we obtain SBFs of size $m = 2.05$ Mbytes with $k = 4$ hash functions.

For routing, push-based BFR requires BF-based content advertisements, while shortest path routing requires to communicate information about the provided content objects and the network topology. In the following, we compare push-based BFR's content advertisement overhead with the communication overhead that is required to calculate the shortest paths. Then, we evaluate all the schemes based on the following performance metrics: 1) normalized communication overhead, 2) average round-trip delay, 3) mean hit distance. We also compare push-based BFR and COBRA in terms of the average memory needed for storing BFs and SBFs. Further, we present results concerning the impact of false positive reports from BFs on push-based BFR routing for different levels of the false positive error. In the following, we discuss results for these metrics.

Table 3.2: False positive error probability under various m/n and k combinations

m/n	k	p_{fpp}
3	2	28.3%
3	2	23.7%
4	3	16.0%
6	4	6.38%
8	5	2.29%

3.4.2 Content Advertisement Overhead

Fig. 3.8 illustrates the total communication overhead needed for propagating content advertisements in push-based BFR for different levels of false positive error probability as well as the required communication overhead for calculating the shortest paths in the shortest path approach. For push-based BFR, we consider four sets of parameters for content advertisement BFs as shown in Table 3.2. From Table 3.2, it is evident the trade-offs between different numbers of hash functions (k), different overhead values per inserted element (m/n), and different values of false positive error probability. As Fig. 3.8 shows, the communication overhead required for calculating shortest paths in shortest path routing is on average approximately three times and five times higher than the communication overhead required for propagating content advertisements in push-based BFR using the GEANT and the grid topologies, respectively. When the grid topology is used, Fig. 3.8 shows that push-based BFR has approximately twice the content advertisement overhead of when GEANT topology is used. From Fig. 3.8, we observe that with the grid topology, shortest path routing requires roughly 3.4 times more communication overhead for calculating the shortest paths than with GEANT topology.

3.4.3 Normalized Communication Overhead

Figs. 3.9a and 3.9b compare results for normalized communication overhead, i.e., the total bandwidth used to forward all Interests and Data packets divided by the number of retrieved Data packets, for GEANT and grid topologies, respectively. From Figs. 3.9a and 3.9b, we observe the very high communication overhead for flooding. This is due to the forwarding of each incoming Interest to all the available faces except the incoming one. We also see from Fig. 3.9a that push-based BFR and shortest path

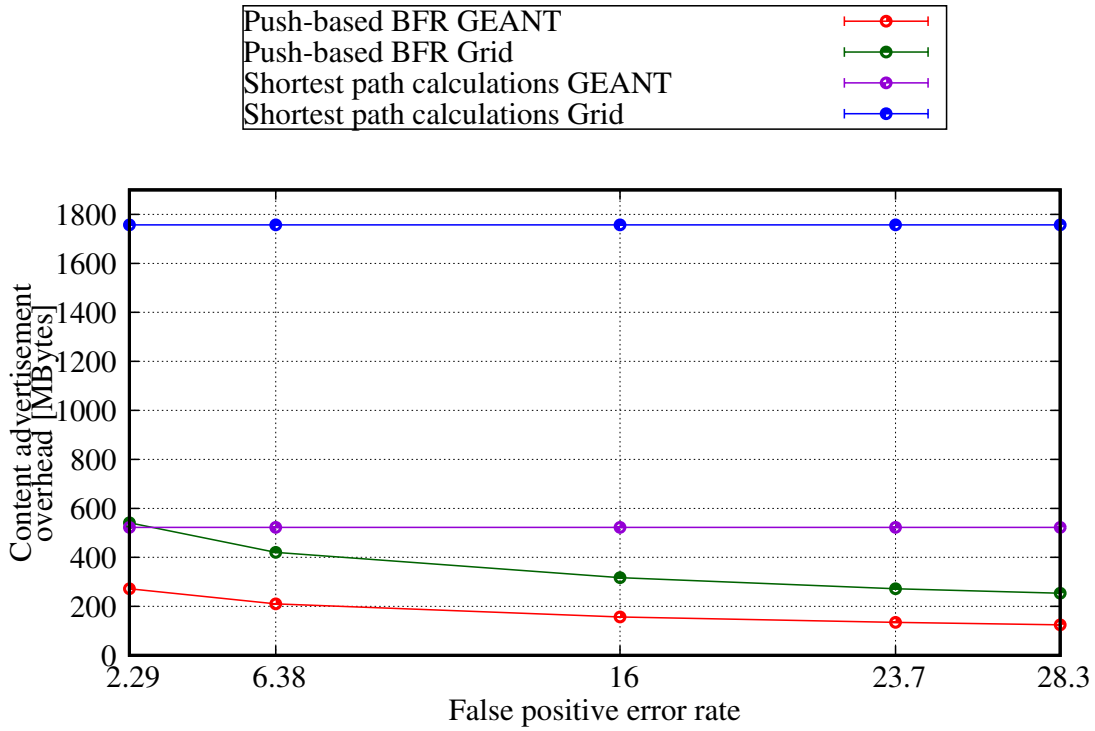
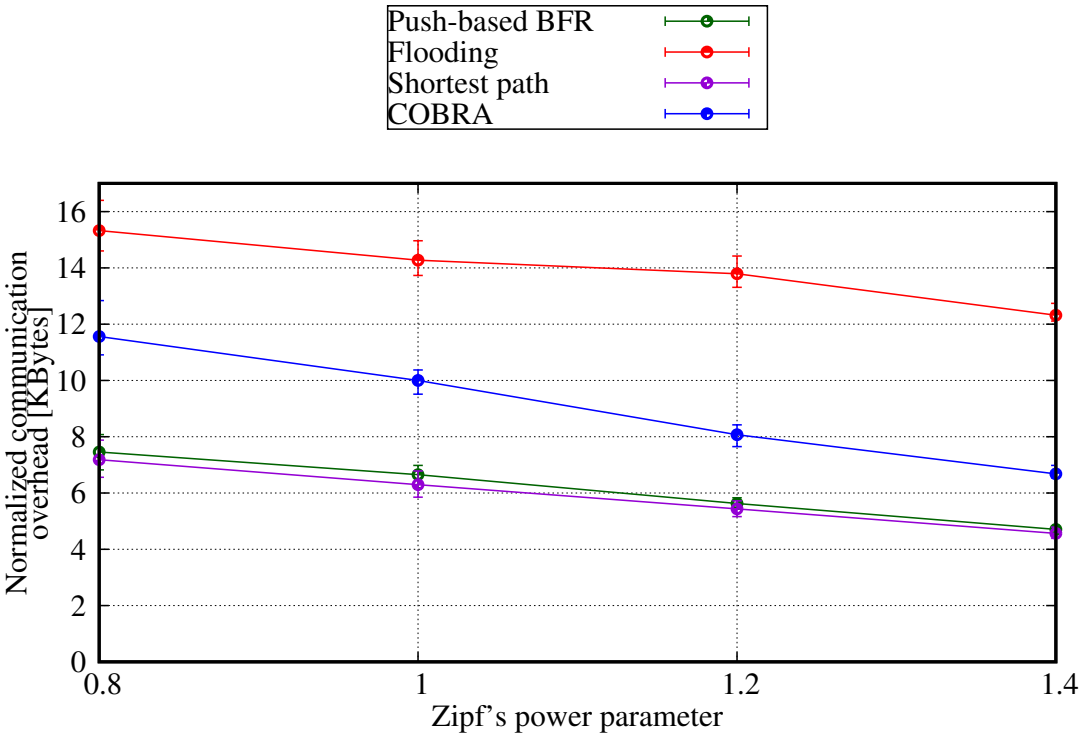
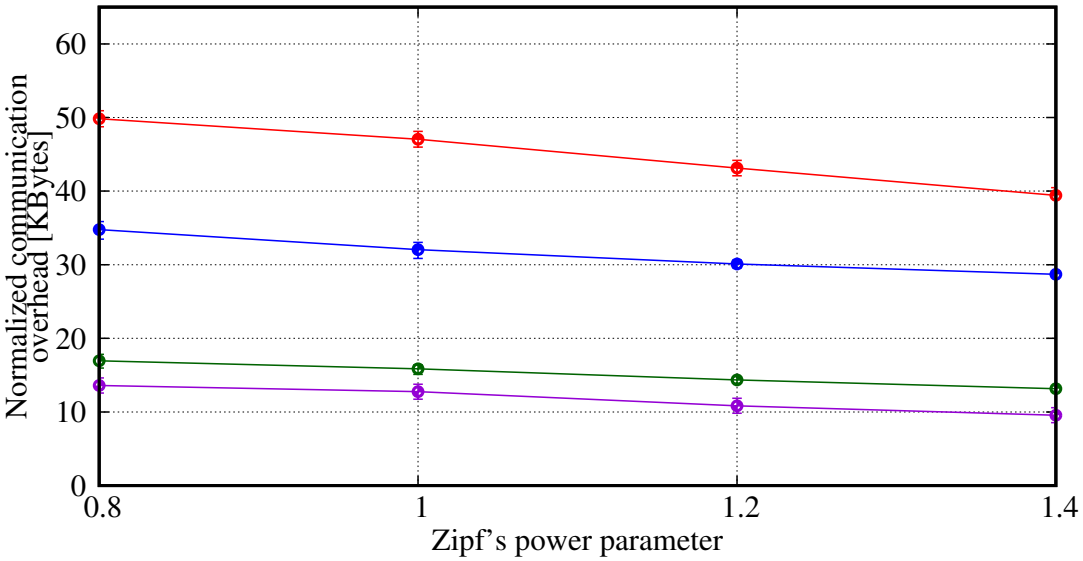


Figure 3.8: A comparison of content advertisement communication overhead vs. false positive probability for the grid and GEANT topologies

routing have quite close normalized communication overhead using GEANT topology. However, when the grid topology is used, Fig. 3.9b shows a larger difference between shortest path routing and push-based BFR in terms of normalized communication overhead because the grid topology is more connected and push-based BFR uses multi-path forwarding strategy, which entails higher communication overhead than forwarding Interests only over the shortest paths. From Figs. 3.9a and 3.9b, we observe that push-based BFR needs much less communication overhead to retrieve a Data packet than COBRA. The reason is that during the learning phase SBFs are empty and COBRA needs to flood the Interests, which incurs significant communication overhead. On the other hand, push-based BFR nodes do not flood the Interests and nodes forward each Interest over the paths en-route to the server(s) that provide the demanded content object. Thus, push-based BFR shows much less normalized communication overhead compared to COBRA. For push-based BFR, when the grid topology is used, Fig. 3.9b shows that the normalized communication overhead for all values of the Zipf's power parameter is approximately twice than when GEANT topology is used. Further, Figs. 3.9a and 3.9b show that flooding and COBRA entail roughly three times higher communication overhead when the grid topology is used



(a) Normalized communication overhead using GEANT topology



(b) Normalized communication overhead using gird topology

Figure 3.9: Results for normalized communication overhead with different values of α

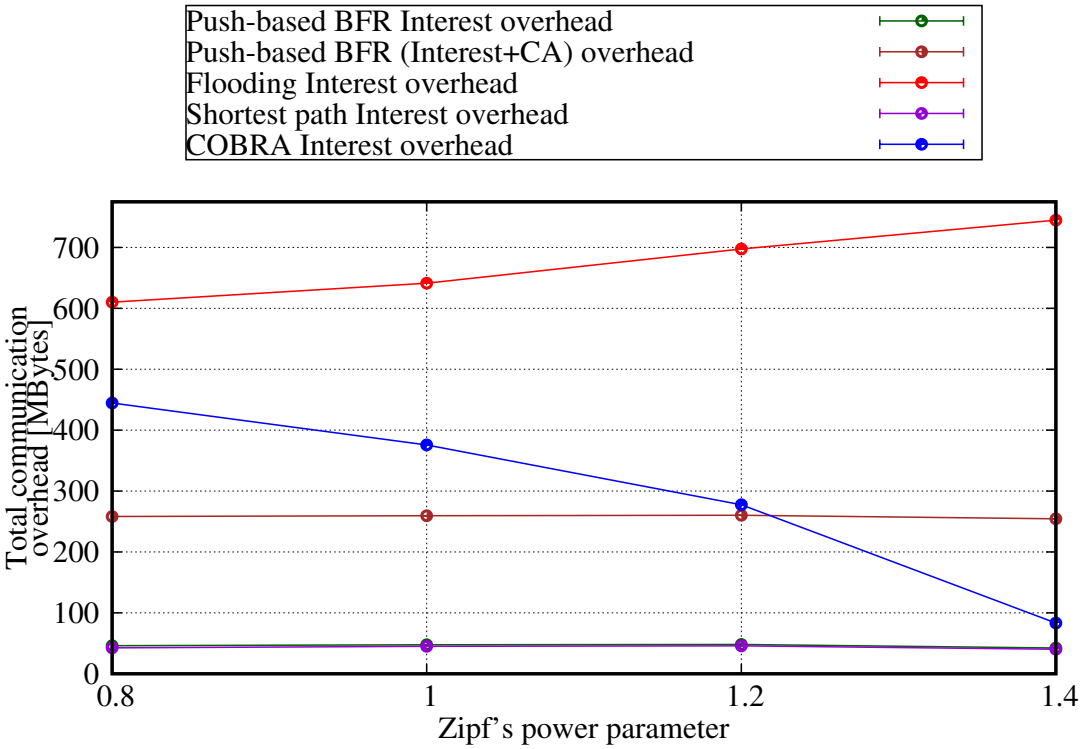
rather than GEANT topology. This behaviour is expected because the grid topology is more connected than the GEANT topology.

3.4.4 Total Communication Overhead for Interests

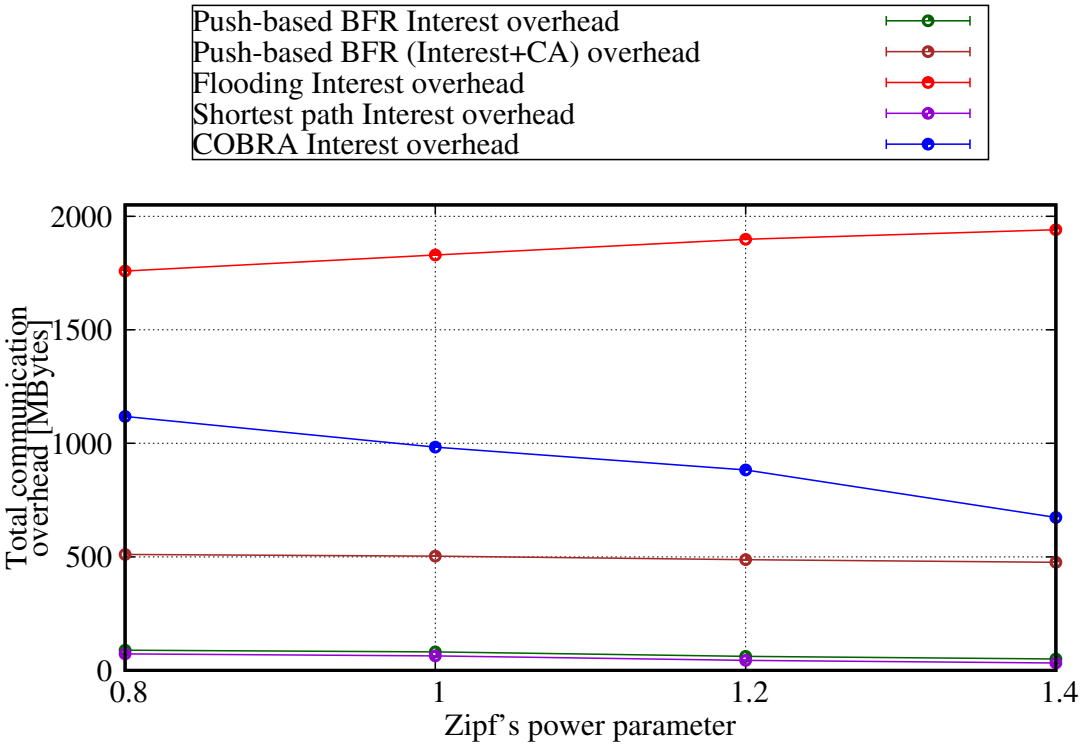
Figs. 3.10a and 3.10b illustrate results in terms of total communication overhead needed for sending Interests for different values of α for GEANT and grid topologies, respectively. Figs. 3.10a and 3.10b show that flooding entails the highest communication overhead due to broadcasting Interests. From Figs. 3.10a and 3.10b, it is clear that the total communication overhead for sending Interests is much higher for COBRA than for push-based BFR. This is because COBRA needs to flood Interests during the learning phase when SBFs are empty. Nevertheless, the gap between the curves of COBRA Interest overhead and push-based BFR Interest overhead is much bigger in Fig. 3.10b than in Fig. 3.10a, because the number of links is much higher in the grid topology than in the GEANT topology. Thus, it leads to much higher communication overhead for forwarding the Interests, specifically during the learning phase when COBRA floods the Interests. For COBRA, if we increase the value of α, we observe that the total communication overhead decreases significantly. The reason is that by increasing the value of α, the cardinality of the set of popular content objects decreases. Thus, the number of Interest floodings during the learning phase also decreases significantly.

In Figs. 3.10a and 3.10b, for push-based BFR, we see results in terms of *(Interest+CA) overhead*, i.e., the sum of the values of total communication overhead needed for sending Interests and the total communication overhead needed for propagating content advertisements in push-based BFR. We still observe a big gap between the curves of COBRA Interest overhead and push-based BFR (Interest+CA) overhead in both Figs. 3.10a and 3.10b. In Fig. 3.10a, we observe this gap at least if $\alpha = 0.8$ or $\alpha = 1$. If $\alpha = 1.4$, Fig. 3.10a shows that COBRA needs much less communication overhead for flooding Interests during the learning phase because the number of popular content objects is much smaller and the topology is tree-like, thus having less number of links.

In Fig. 3.10b, we observe a bigger gap between the curves of COBRA Interest overhead and push-based BFR (Interest+CA) overhead than the gap between these curves in Fig. 3.10a. This is due to the higher impact of Interest floodings in COBRA during the learning phase, with a more connected topology.



(a) Total communication overhead using GEANT topology



(b) Total communication overhead using gird topology

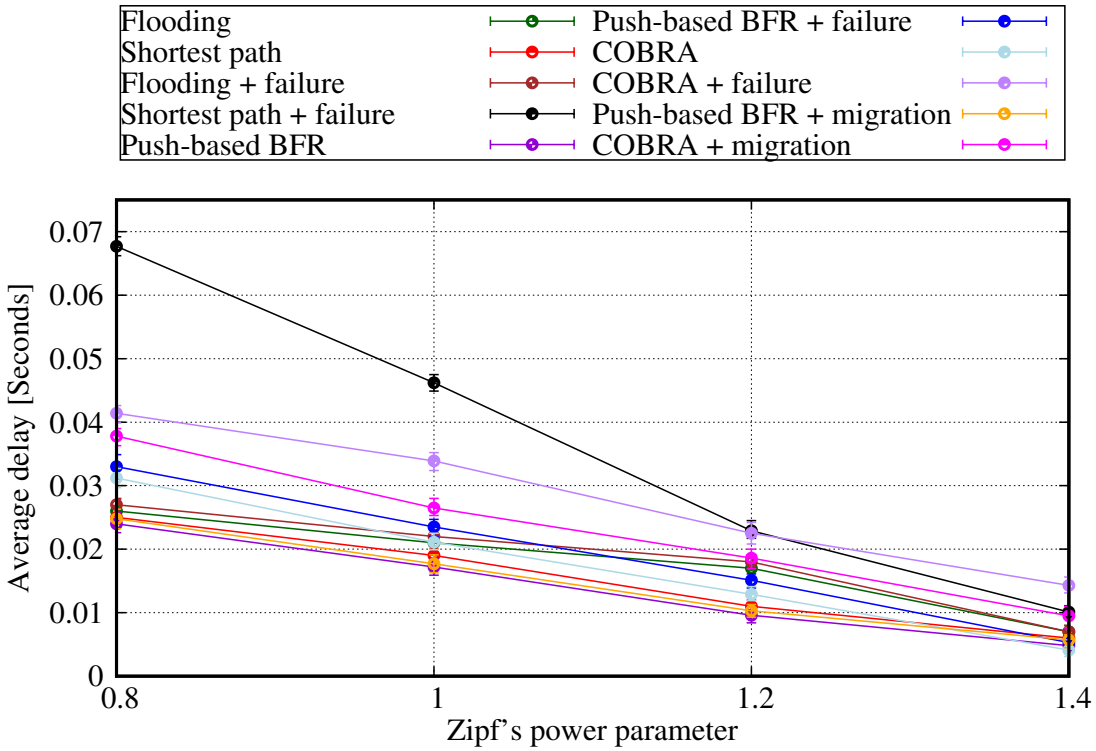
Figure 3.10: Results for total communication overhead for different values of α

3.4.5 Average Round-trip Delay

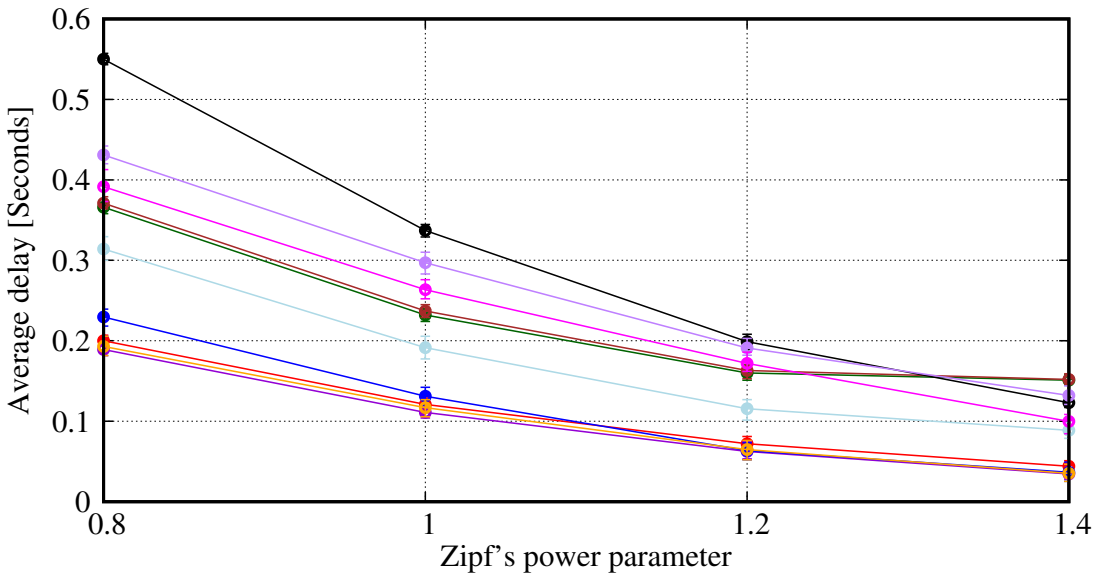
We evaluate the performance of all the schemes under comparison in terms of average round-trip delay, i.e., the average delay from the time instant clients send Interests until the time they retrieve the demanded content objects. To better show the behaviour of all the considered schemes in the presence of topology changes, we also measure the average round-trip delay in the presence of link failures for all the schemes. We schedule three link failures at time instants 5′000 s, 15′000 s, and 25′000 s. These links recover at time instants 10′000 s, 20′000 s, and 30′000 s, respectively.

Figs. 3.11a and 3.11b illustrate the results in terms of average round-trip delays. From Figs. 3.11a and 3.11b, we observe that flooding shows the highest delay in the absence of link failures. The reason is that flooding all the Interests creates bottlenecks and results in high delays. The shortest path approach has a lower average delay compared to flooding because it forwards each Interest only through the face that has the shortest path to the origin server. This is not always efficient as the shortest path is not always the "best" path. In [14], the authors show that the "best" path is the one with the highest throughput or the least congested path in other words. Push-based BFR benefits from multi-path forwarding and hence transmits Interests through all the faces that the demanded content object can be reached with high probability. When the shortest paths are congested, push-based BFR also exploits longer, but less congested paths for sending the Interests and thus performs better than shortest path routing in terms of delay. From Figs. 3.11a and 3.11b, we observe that push-based BFR outperforms COBRA without link failures. The reason is that when COBRA routers insert new Data names into the SBFs, they decrease the values of some randomly selected SBF counters, which could lead to the removal of some route traces from the SBFs. Therefore, in contrast to push-based BFR, COBRA does not always use all the available paths to the origin server of the demanded content object.

In the presence of link failures, Figs. 3.11a and 3.11b confirm the resilience of flooding to the link failures because it broadcasts the Interests and forwards them over all the paths. These figures show that push-based BFR is also resilient to link failures in terms of delay. This is due to the fact that push-based BFR benefits from the existence of multiple paths towards origin servers and does not forward the Interests over a single path. From Figs. 3.11a and 3.11b, we also observe that shortest path routing is the less resilient approach to link failures. This is because it always relies on a single path and forwards the Interests over this path to the origin server of the demanded content



(a) Average round-trip delay using GEANT topology



(b) Average round-trip delay using gird topology

Figure 3.11: Results for average round-trip delay

objects, while a link failure might occur on that path. In the presence of link failures, both push-based BFR and COBRA routing protocols avoid sending an Interest through the path over which a link has failed. However, push-based BFR forwards the Interest over the rest of the paths towards the server that provides the demanded content object, while with COBRA, nodes do not always benefit from all the paths towards the demanded content objects. We see in Fig. 3.11b a smaller impact of link failures on push-based BFR's performance in terms of average round-trip delay than in Fig. 3.11a. This means that push-based BFR is more resilient to link failures when the topology is more connected.

We also examine the performance of push-based BFR and COBRA with content migration. We schedule a random number of content migration events (between 2 to 4 content migration events) between randomly selected servers at random time instants. We observe from Figs. 3.11a and 3.11b that with push-based BFR, content migration events have a slight impact on the average round-trip delay. The reason is that, when a content migration happens and push-based BFR is used, servers immediately propagate new CAI messages to inform clients and routers about this event, thus clients and routers update routes once they receive new CAI messages. However, with COBRA, clients and routers are unaware of content migration events until they detect Interest retransmissions. Therefore, for COBRA, we see from Figs. 3.11a and 3.11b a much higher impact of content migration events on the average round-trip delay.

3.4.6 Robustness to Topology Changes

Fig. 3.12 compares the performance of all the considered schemes in terms of the impact of link failures on the percentage of unsatisfied Interests for different values of α. Fig. 3.12 shows that all the Interests are satisfied in the presence of link failures when flooding is used, with both GEANT and gird topologies, because the flooding approach broadcasts the Interests and does not rely only on the paths on which links have failed. Using push-based BFR, the maximum rate of unsatisfied Interests is only 0.93% with GEANT topology, while all Interests are satisfied with the grid topology. This is attributed to the fact that the grid topology is more connected. Therefore, push-based BFR is more robust to link failures using the grid topology. From Fig. 3.12, we can see that the maximum rate of unsatisfied Interests for shortest path routing is approximately 6.4%, when GEANT topology is used. The performance of shortest path routing degrades in the presence of link failures because it always relies on the

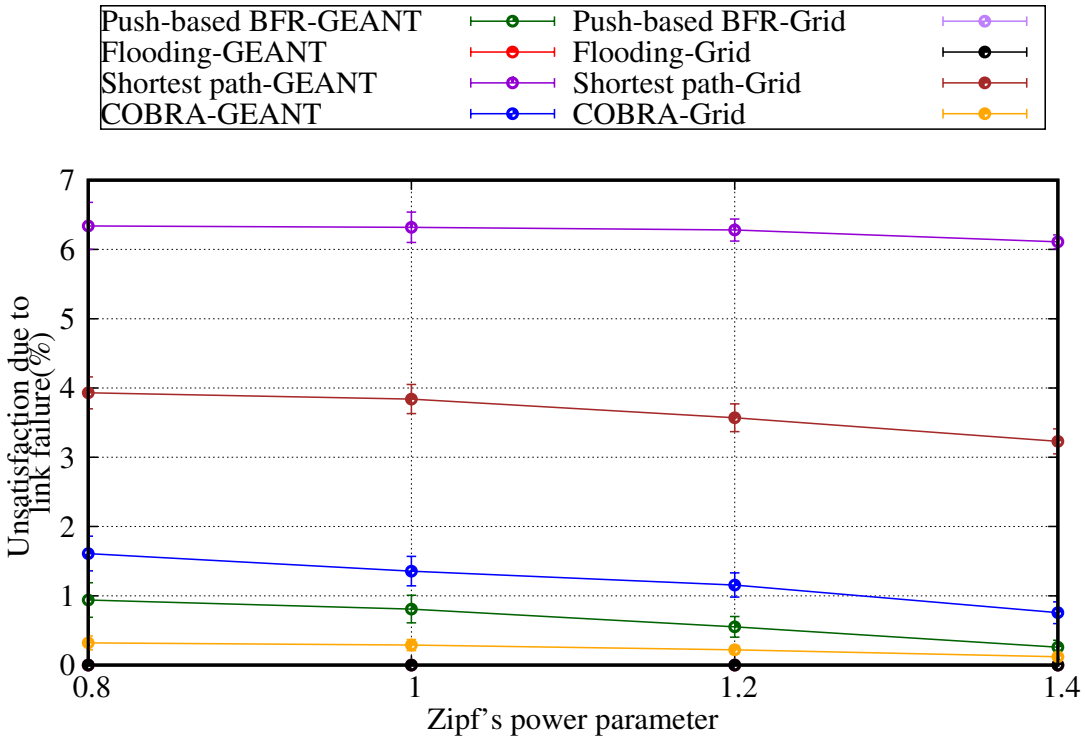
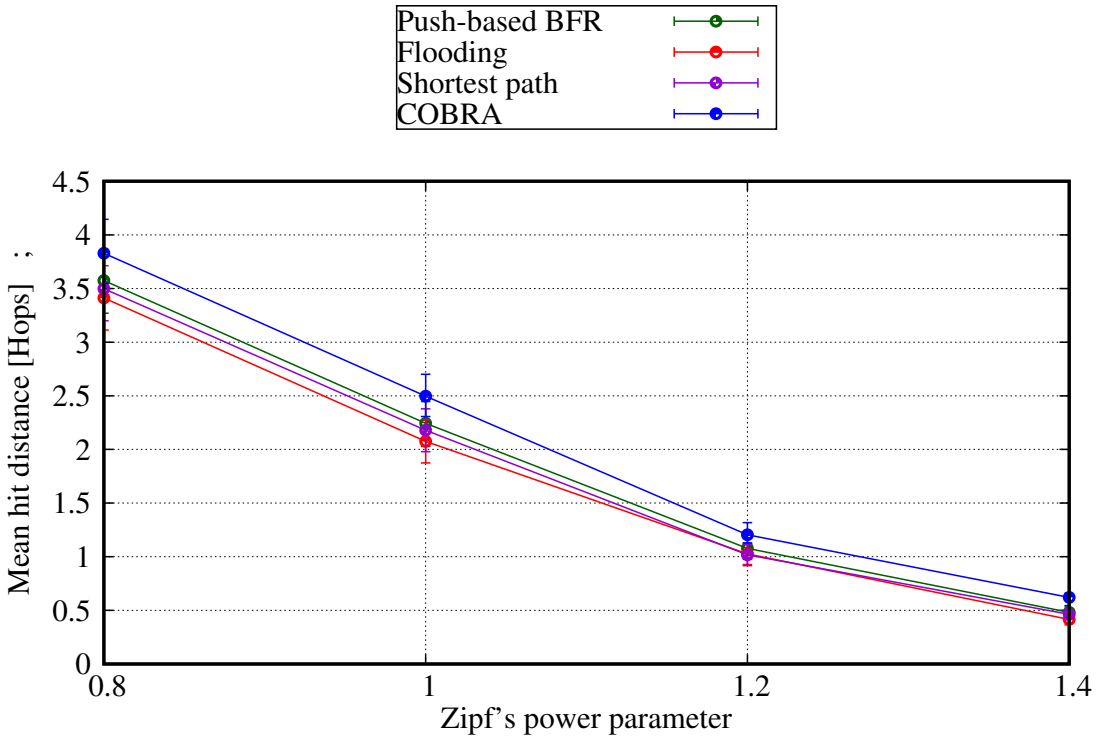


Figure 3.12: Results for the impact of link failures on Interest unsatisfaction for different values of α

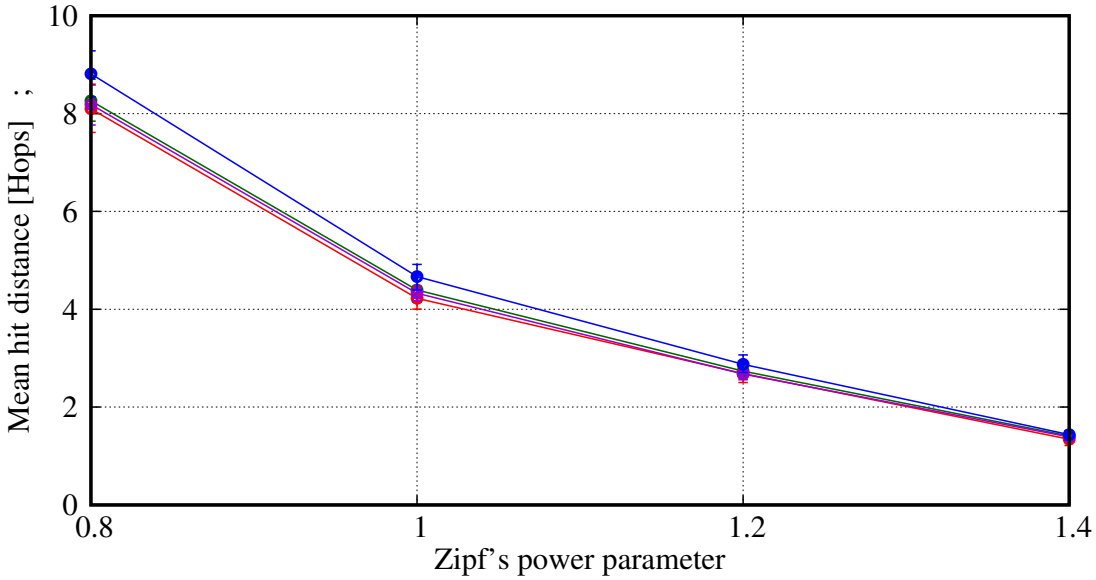
shortest path towards the origin server of the demanded content object on which links might fail. Nevertheless, with the grid topology, shortest path routing shows lower rates of unsatisfied Interests, i.e., a maximum of 3.93%, which is due to the higher connectivity of the grid topology. Fig. 3.12 shows that with both GEANT and grid topologies, push-based BFR is more robust to link failures than COBRA because using COBRA, routers randomly remove some of the route traces from the SBFs, while push-based BFR always benefits from all available paths to the origin server(s) of the demanded content objects.

3.4.7 Mean Hit Distance

We present the results in Figs. 3.13a and 3.13b concerning the mean hit distance, i.e., the mean path length an Interest message requires traveling to reach the demanded content object, for GEANT and grid topologies, respectively. The first transmission of each Interest in the network has to reach the server that provides the demanded content object. However, subsequent transmissions of the Interest can be retrieved



(a) Mean hit distance using GEANT topology



(b) Mean hit distance using gird topology

Figure 3.13: Results for mean hit distance for different values of α.

Table 3.3: Average memory needed for storing routing information at each node with different false positive error rates.

p_{fpp} (%)	Average allocated memory per node (KBytes)		
	COBRA (GEANT)	COBRA (Grid)	Push-based BFR (Any topology)
28.30%	6235.392	14725.121	32.072
23.70%	7111.619	16783.365	36.579
16.00%	9052.344	21370.883	46.561
6.38%	13594.010	32880.641	69.921
2.29%	18655.232	44042.242	95.954

from routers' caches. Figs. 3.13a and 3.13b show that the flooding approach has a slightly better performance for $\alpha = 0.8$ and $\alpha = 1$. However, for $\alpha = 1.2$ and $\alpha = 1.4$, flooding and push-based BFR perform approximately equal in terms of mean hit distance. We see from Figs. 3.13a and 3.13b smaller values of mean hit distance for push-based BFR than for COBRA, which does not exceed 0.45 hops.

3.4.8 Average Memory Needed for Storing Routing Information

Table 3.3 compares push-based BFR and COBRA in terms of the average memory space that each node needs to store routing information. In push-based BFR, routing information consists of the content advertisement information that servers propagate. Thus, the storage overhead for routing information is not related to the structure of the topology. However, in COBRA, routing information consists of the route traces stored in the SBFs. In COBRA, each node stores as many SBFs as the number of its interfaces. Therefore, the number of SBFs is directly proportional to the number of links. Thus, the more connected the topology is, the higher is the storage overhead COBRA requires to store SBFs. To better understand this, we compare the average memory needed for storing SBFs in COBRA for GEANT and grid topologies in Table 3.3. For COBRA, columns 2 and 3 of Table 3.3 show that in the grid topology each node needs approximately 2.36 times more memory to store SBFs than in the GEANT topology. Nevertheless, we observe a significant difference between the values of columns 2 and 4. This means that even if the GEANT topology is used, push-based BFR needs several magnitudes less memory space for storing routing information than COBRA. Therefore, when nodes have restricted memory capacity (e.g., sensors in IoT scenarios with constrained nodes), it is more appropriate to use push-based BFR than COBRA.

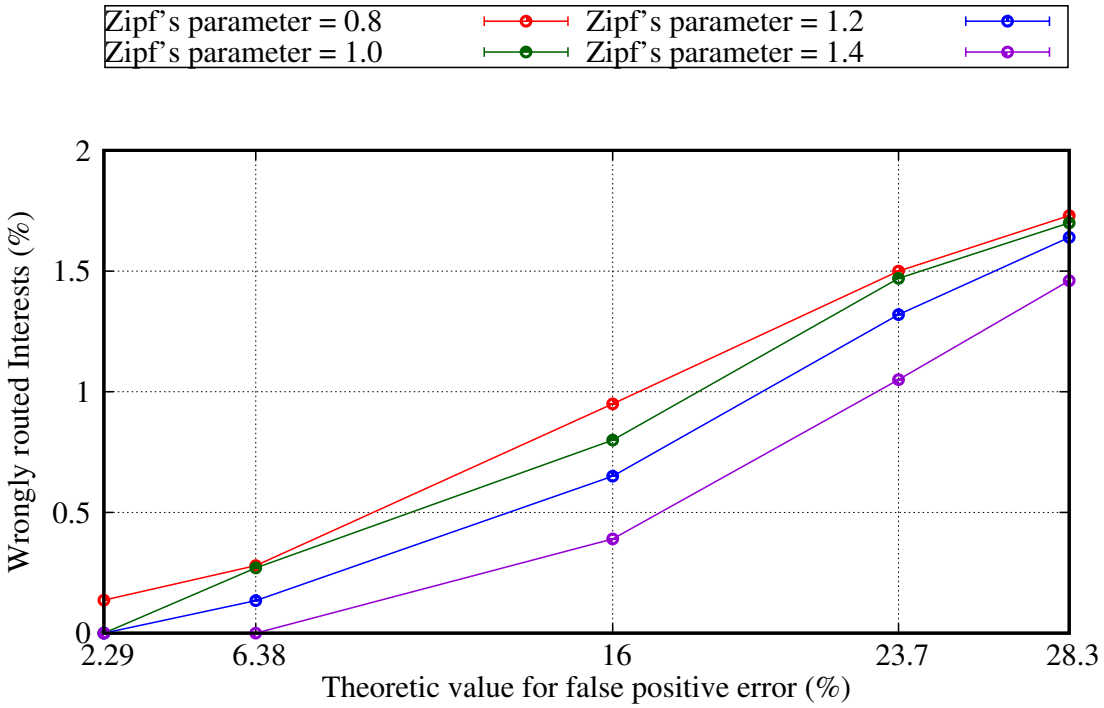


Figure 3.14: Impact of false positive reports on push-based BFR routing for different values of p_{fpp} and α

3.4.9 Impact of False Positive Errors on Routing

We present results concerning the impact of false positive errors on push-based BFR operation in terms of percentage of Interests that have been routed towards both correct and wrong origin servers for different values of p_{fpp} shown in Table 3.2. For COBRA, false positive errors are not the only cause of routing Interests towards wrong origin servers, but Interests might reach wrong origin servers due to Interest floodings.

Fig. 3.14 shows that the higher the probability of false positive error is, the higher are the number of Interests that not only have been routed towards correct origin servers, but have reached wrong origin servers as well. Further, Fig. 3.14 shows the impact of increasing the value of α on the percentage of Interests that are also routed towards wrong origin servers. We note that when the value of α is higher, a smaller set of content objects are popular and this results in measuring less false positive reports in practice. We observe the highest impact of false positive reports on push-based BFR routing for $p_{fpp} = 28.3\%$ and $\alpha = 0.8$, when only 1.73% of Interests are routed also towards wrong origin servers. Note that all the Interests are satisfied in the presence

of false positive reports and the only practical impact of false positive reports is that a very small number of Interests reach wrong origin servers, i.e., the origin servers that do not provide the demanded content objects, while all the Interests are routed towards correct origin servers, i.e., the origin servers that provide the demanded content objects.

3.5 Conclusions

In this Chapter, we presented push-based BFR, a BF-based, fully distributed, content-oriented, and topology agnostic routing protocol at the intra-domain level for NDN. Push-based BFR is based on the propagation of content advertisements from origin servers using BFs. We compared push-based BFR with flooding, shortest path, and COBRA. Push-based BFR outperforms flooding, shortest path routing, and COBRA in terms of communication cost, and average round-trip delay. In terms of robustness to topology changes, push-based BFR strongly outperforms the shortest path approach. In contrast to schemes based on shortest path routing, push-based BFR does not require any auxiliary routing protocols for calculating the best paths. Therefore, push-based BFR entails significantly less content advertisement overhead than the shortest path protocol. Push-based BFR outperforms COBRA in terms of the average memory needed for storing routing information. Therefore, it is more appropriate to use push-based BFR than COBRA when network nodes have restricted memory capacity (e.g., IoT scenarios with sensor networks). In the next Chapter, we focus on reducing the communication and storage overhead of BF-based content advertisements.

4

Pull-based Bloom Filter-based Routing

4.1 Introduction

Chapter 3 presented push-based BFR as a routing protocol for NDN, which is fully distributed, topology oblivious, and fully content-oriented without any dependency in IP addresses. In Chapter 3, we showed that using BFs, we can compress content advertisements. In this Chapter, we attempt to answer RQ 2 of the thesis Introduction that is related to further reducing the required bandwidth and storage resources for BF-based content advertisements.

Although there are too many content objects available, users are not interested in retrieving all the content objects. In contrast, in real-world scenarios, most of the users are usually interested in downloading popular content objects. Thus, it is not necessary to advertise all the names of the entire content universe (i.e., all provided content objects). Therefore, we design a BF-based routing protocol, which only advertises the names of the requested content objects.

We propose *pull-based BFR* [51] as a BF-based routing protocol that primarily informs

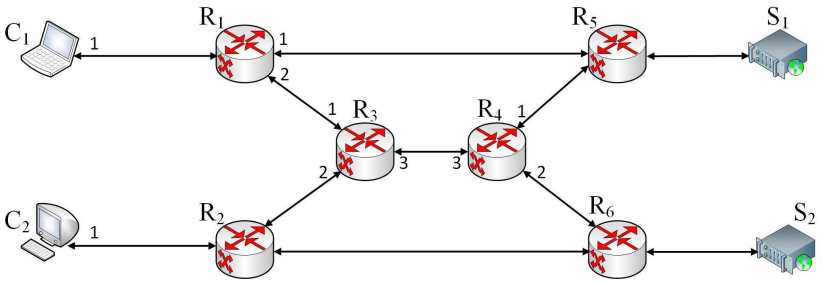
the servers about the content object names that are requested. For this purpose, when clients issue Interest messages, they store their Interest message names into BFs and propagate the BFs using Content Advertisement Request (CAR) messages. When routers receive CAR messages, they aggregate them and proceed with a CAR propagation process so that CAR messages reach the servers. When a server receives CAR messages, it uses the BF-based information stored in these messages to populate a list of the requested content object names. Then, the server compresses the list of the requested content object names via a BF and advertises this BF using Content Advertisement (CA) messages. Note that routers have to aggregate the received CA messages and proceed with propagating them so that clients receive CA messages. When clients receive the CA messages, they can populate their FIBs and can forward their pending Interest messages to retrieve their requested content objects. In this Chapter, we propose novel and practical BF aggregation methods to aggregate CAR messages as well as CA messages.

In the following, we describe the proposed pull-based BFR, and we compare its performance with push-based BFR using different content universe sizes. Our performance evaluation results show that pull-based BFR has several advantages compared to push-based BFR, namely: 1) significantly less bandwidth consumption for propagating content advertisements, 2) significantly less storage space requirements for clients and routers to store content advertisements, 3) better average round-trip delay when fewer content objects are popular, and 4) more robustness to false positive reports from BFs.

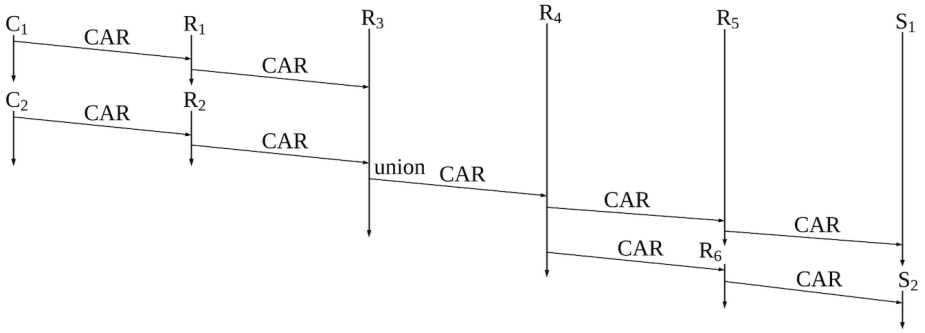
The rest of this Chapter is organized as follows. Section 4.2 describes the proposed pull-based BFR protocol. Then, Section 4.3 presents the performance evaluation of the proposed pull-based BFR and compares it with push-based BFR and FaR [68] approaches. Finally, Section 4.4 concludes the Chapter.

4.2 Pull-based Bloom Filter-based Routing

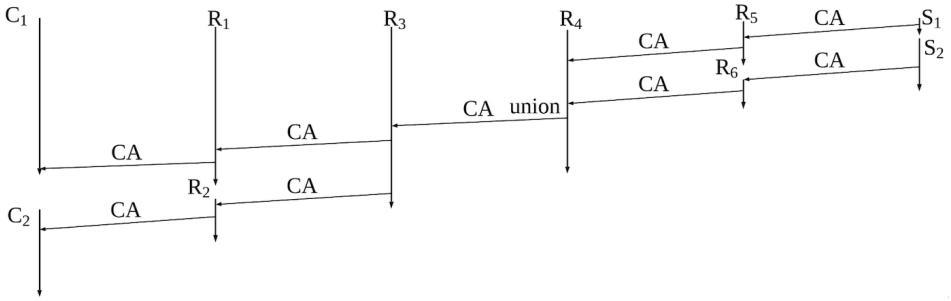
The rationale behind designing a pull-based BFR method is to advertise only the demanded content objects. When servers only advertise the demanded content objects, it is expected that: 1) a significant amount of bandwidth will be saved, and 2) other network nodes (clients and routers) will need significantly less memory space to store content advertisement information. This content advertisement strategy can



(a) A topology for describing pull-based BFR



(b) CAR transmissions



(c) CA transmissions

Figure 4.1: CAR and CA transmissions

resolve scalability issues of push-based BFR, as in push-based BFR servers advertise all the file names of the content universe. The main difficulty arising from advertising only the demanded content objects is that servers do not know a priori which content objects will be demanded. To overcome this problem, in pull-based BFR, we follow a BF-based strategy to inform the servers about the demanded file names, which we will explain in the next sub-section.

4.2.1 Pull-based BFR's Operation

Content advertisement in pull-based BFR is performed in two consecutive phases: 1) clients and routers use a BF-based strategy to inform the servers about the demanded file names, and 2) servers proceed with the advertisement of these names using CA messages. Upon reception of CA messages, clients and routers store the content advertisement information and populate the FIBs for pending Interests to route them. To summarize, pull-based BFR's operation is done in three stages: 1) pulling content advertisements, 2) content advertisement, and 3) FIB population and content retrieval.

Let us explain our BF-based method of informing servers about the demanded file names with the help of Fig. 4.1, where Fig. 4.1a depicts a topology for describing CAR and CA transmissions, Fig. 4.1b illustrates CAR transmissions, and 4.1c shows CA transmissions.

In Fig. 4.1a, we assume that client C_1 issues Interest I_1 to retrieve a segment of file name N_1 under the following conditions: 1) there is no FIB entry for N_1 or a name prefix of it, and 2) there is no stored content advertisement BF that contains N_1 or a name prefix of it. Thus, client C_1 avoids forwarding Interest I_1 and keeps it as pending. Nevertheless, client C_1 informs the servers that file name N_1 is demanded to pull the content advertisement information for it. For this purpose, client C_1 creates a BF, which contains file name N_1 as well as all its name prefixes and creates a CAR message of type Interest called CAR_{C_1} with name $/CAR/C_1/sequenceNumber$ that encapsulates the BF. Then, client C_1 broadcasts CAR_{C_1} to inform the servers about the demanded file names and to pull the needed content advertisements.

When a router receives a CAR message, it waits for an *aggregation threshold time* δ to receive other CAR messages issued by other clients. Assume that client C_2 issues Interest I_2 to demand a segment of file name N_2 for which no FIB entry and no content advertisement information is available. Thus, client C_2 broadcasts a CAR message called CAR_{C_2} with name $/CAR/C_2/sequenceNumber$ carrying a BF that contains file name N_2 as well as all its name prefixes. If router R_3 receives the CAR messages of clients C_1 and C_2, within a time interval δ, it forwards CAR_{C_1} and CAR_{C_2} over faces 1 and 2, respectively. At the same time, router R_3 forwards the aggregation of CAR_{C_1} and CAR_{C_2} over face 3. To aggregate CAR_{C_1} and CAR_{C_2}, router R_3 makes a union of their BFs and puts the resulting BF into a

new CAR message with name $/CAR/aggregated/R_3/sequenceNumber$. Then router R_3 forwards this message over face 3. When router R_3 forwards message $/CAR/aggregated/R_3/sequenceNumber$ over face 3, router R_3 updates the out-records of both messages CAR_{C_1} and CAR_{C_2} by adding face 3 to record that both these messages have been forwarded over face 3. Further, router R_3 will not use message $/CAR/aggregated/R_3/sequenceNumber$ in future aggregations, because the third name component specifies that this message is created by router R_3 itself and has not been received from other nodes. Routers R_4, R_5, and R_6 follow the same forwarding process for CAR messages. Nodes make use of a *sequence number counter* for calculating the sequence numbers of CAR messages.

To permit BF union operations, we assume that all nodes create the BFs of the CAR messages with the same size, and that they generate the hash functions using a universal seed, i.e., all nodes use the same set of hash functions for BFs. In Eq. (2.1), if we assign a constant value to m and we specify the value of p, we will derive the maximum optimal value for n, which estimates the maximum number of requested file names that can be inserted into the BF. It is not a problem that all nodes use a universal seed to generate the hash functions for the BFs of all CAR messages, as all nodes can use a well-known word, e.g., *NDN* as the universal seed to generate hash functions.

When servers S_1 and S_2 receive a CAR message, they check all the produced file names against the BF of the received CAR message (we assume that servers have multi-core processors and can check multiple names against multiple BFs in parallel. Thus, this does not create a performance issue). The file names that exist in the BF of the CAR message are the demanded file names that should be advertised. Thus, both servers S_1 and S_2 first create a list of these file names called *toBeAdvertisedList* and then a BF called *toBeAdvertisedBF* with size equal to that of the received CAR message's BF. When a server notes that a produced file name exists in the BF of the CAR message, it inserts the file name into the BF *toBeAdvertisedBF*. Then, the server creates a CA message, from type Interest with name prefix $/CA/serverID/sequenceNumber$ carrying the *toBeAdvertisedBF*. The server broadcasts the CA message to the network to advertise the demanded content object and not to demand any content objects. In our example, if router R_4 receives the CA messages of servers S_1 and S_2, namely, CA_{S_1} and CA_{S_2}, which have the names $/CA/S_1/sequenceNumber$ and $/CA/S_2/sequenceNumber$, respectively, within a time interval δ, Router R_4 forwards CA_{S_1} and CA_{S_2} over faces 1 and 2, respectively. Router R_4 aggregates CA_{S_1} and CA_{S_2}

unioning their BFs and places the resulting BF into an aggregated message, which has the name $/CA/aggregated/R_4/sequenceNumber$ and forwards this message over face 3.

When clients C_1 and C_2 receive the CA message, they can populate their FIBs for name prefixes N_1 and N_2, which allows them to route Interests I_1 and I_2. When routers receive Interests I_1 and I_2 from the clients, they also populate the FIBs using the stored CA messages and continue routing the Interests until the demanded content objects are retrieved.

4.2.2 Bloom Filter Aggregation

If a router makes a union of the BFs BF_1 and BF_2, which are not subset or equal to each other, i.e., $(BF_1 \nsubseteq BF_2) \wedge (BF_2 \nsubseteq BF_1)$, the number of 1 bits in the bit vector of the resulting BF BF_{union} will be greater than the number of 1 bits in each of BF_1 and BF_2. Thus, if routers do not stop unioning BFs that are not subset or equal to each other, at some point all the bits of the bit vector of the resulting BF will be set to 1. Such a BF does not function properly because it falsely claims that it contains all the existing names. Therefore, routers should stop unioning the BFs of both CAR and CA messages according to the maximum capacity of BFs. As we explained before, we consider a constant size of m and a probability of false positive error p for the BFs of CAR messages. Then, using Eq. (2.1), we calculate n, which is the maximum capacity of the BF.

To describe the BF aggregation process, in Fig. 4.1a, we assume that router R_3 receives two CAR messages CAR_1 and CAR_2 from routers R_1 and R_2, respectively. CAR_1 and CAR_2 contain two BFs BF_1 and BF_2, which have inserted element counts $|BF_1|$ and $|BF_2|$, respectively. If router R_3 wants to aggregate BF_1 and BF_2, it first checks whether BF_1 and BF_2 are identical. For this purpose, router R_3 makes an XOR of the bit vectors of BF_1 and BF_2. If all the bits of the resulting bit vector are zero, BF_1 and BF_2 are identical. In such a case, there is no need to make a union of them. The second check is to examine whether the following proposition is true $(BF_1 \subset BF_2) \vee (BF_2 \subset BF_1)$. For this purpose, router R_3 calculates $BF_{intersection} = BF_1 \cap BF_2$. If the resulting bit vector is identical with the bit vector of BF_1, it means that $BF_1 \cup BF_2 = BF_2$. In this case, again router R_3 does not need to calculate the union of BF_1 and BF_2. If $(BF_1 \nsubseteq BF_2) \wedge (BF_2 \nsubseteq BF_1)$, then router R_3 makes a union of BF_1 and BF_2. In this case, if $BF_{union} = BF_1 \cup BF_2$ and $BF_{intersection} = BF_1 \cap BF_2$, theoretically we have

$|BF_{union}| = |BF_1| + |BF_2| - |BF_{intersection}|$. However, practically it is not possible to calculate $|BF_{intersection}|$, precisely. Therefore, router R_3 sets $|BF_{union}| = |BF_1| + |BF_2|$, which is a conservative upper bound. If $|BF_1| + |BF_2| < n$, router R_3 will aggregate BF_1 and BF_2. Otherwise, router R_3 avoids aggregating these BFs.

4.2.3 The Impact of False Positive Errors on Pull-based BFR's Operation

The impact of false positive errors on the operation of pull-based BFR should be considered in two cases: 1) if servers check the produced file names against the CAR messages BFs, 2) if clients or routers check the pending Interest names against the CA messages BFs. Consider in Fig. 4.1a that server S_1 receives a CAR message carrying a BF, which contains names N_1 and N_2. If server S_1 checks file name N_3 against the received BF and the BF gives a false positive report, server S_1 will insert name N_3 into the BF of the CA message $/CA/S_1$ and advertises this message. Therefore, the CA message $/CA/S_1$ advertises file name N_3, which has not been demanded. This is not a problem because it is guaranteed that no false negative errors happen using BFs, and, therefore, servers advertise the produced file names that are demanded anyways.

Let us again examine Fig. 4.1a to discuss the impact of false positive reports from the BFs of CA messages, when clients or routers check the Interest names against these BFs for FIB population and routing purposes. In Fig. 4.1a, we assume that router R_4 checks the name N_i for Interest i against the BF of CA message CA_{S_1} issued by server S_1. If the BF gives a false positive report, router R_4 will forward the Interest i over face 1. Consequently, Interest i will be routed towards a wrong server, i.e., server S_1. When server S_1 receives Interest i, it sends back a "No Data" Nack message [87] to inform router R_4 that server S_1 does not store the Data that Interest i requests. When router R_4 receives the Nack message, it will remove from the FIB the incorrect next hop information corresponding to name N_i. Further, if Interest i is not satisfied yet, router R_4 will send a CAR message containing N_i to receive the correct routing information.

4.3 Performance Evaluation

We consider the following metrics for assessing the performance of pull-based and push-based BFR: 1) content advertisement overhead, 2) storage space requirements for storing routing information, and 3) the impact of false positive errors of BFs on

routing. We also evaluate the performance of push-based and pull-based BFR in terms of average round-trip delay. Further, we compare push-based and pull-based BFR with FaR [68] in terms of average round-trip delay to have a more complete analysis. We implemented all protocols in ndnSIM2.1 [56].

4.3.1 Simulation Settings

To compare the performance of the protocols under comparison, we use the GEANT topology [2, 50] illustrated in Fig. 3.6. The topology is built by randomly placing 10 servers and 50 clients in the GEANT topology, which connects 40 routers. Thus, the resulting topology consists 100 nodes. We assume that the content popularity follows Zipf-Mandelbrot distribution (we showed the Zipf-Mandelbrot probability distribution formula in Eq. (3.2)). We consider the values of α in the $[0.6, 2]$ interval. We use a URL dataset extracted from real HTTP request traces [26]. We assume that the content universe has $100,000$ file names and that each is divided into 100 segments. Therefore, there are 10^7 unique segments. For the BFs of CAR and CA messages, we set $m = 716$ Bytes and $P_{fpp} = 0.0638$. Recall, that m is the BF's bit vector size and P_{fpp} represents the false positive probability. Hence, using Eq. (2.1), the maximum value of n will be 1000.

4.3.2 Content Advertisement Overhead

For pull-based BFR, Fig. 4.2 shows the content advertisement overhead, i.e., the total communication overhead required for forwarding CAR and CA messages in terms of *forwarding rate of routing messages*, which is defined as $\frac{1}{\delta}$. Higher *forwarding rate of routing messages* results in more frequent forwarding of CAR and CA messages, i.e., less aggregation of CAR and CA messages. We set the δ values in the $[0.1, 6.4]$ interval measured in milliseconds. This results in the forwarding rate of routing messages in the $[0.16, 10]$ interval in terms of kilohertz. From Fig. 4.2, we observe that for pull-based BFR, the content advertisement overhead increases by increasing the forwarding rate of routing messages, for all α values. For push-based BFR, the total communication overhead needed for content advertisements depends on the content universe size, because servers advertise all the file names they produce. However, in pull-based BFR, servers do not advertise the file names that are not demanded. The number of popular files is controlled by the value of α (higher α means less content objects are requested). We observe from Fig. 4.2 that for pull-based BFR, the

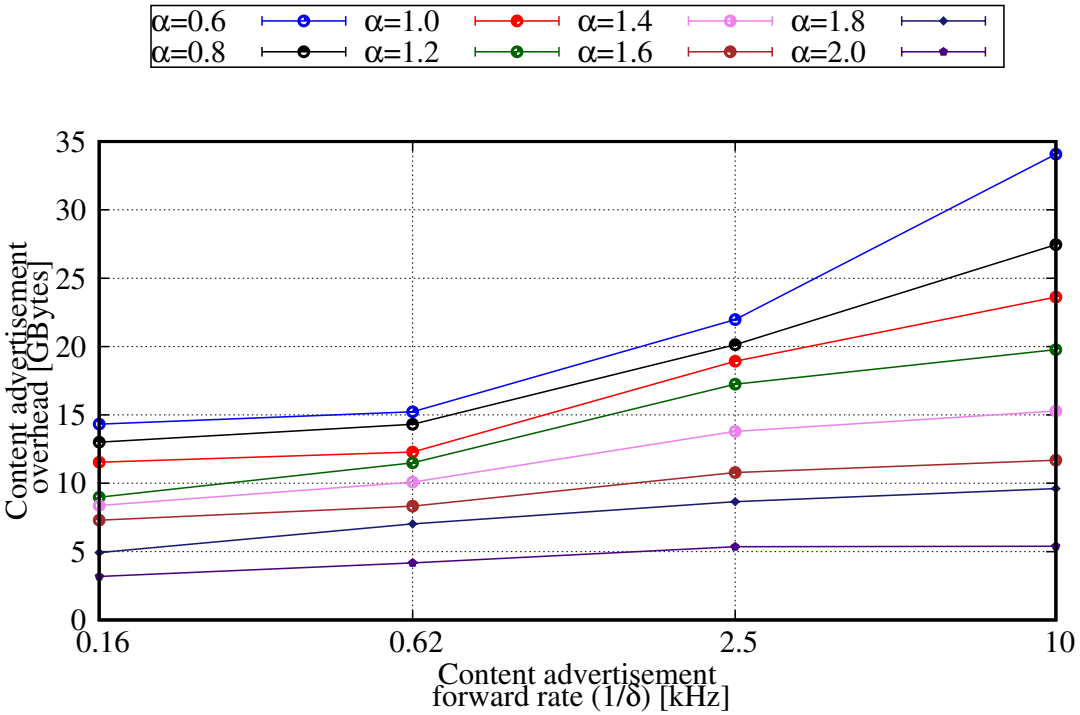


Figure 4.2: Results for content advertisement overhead for different values of δ

communication overhead needed for content advertisements significantly decreases with higher α values. This is due to the fact that when the value of α increases, less content objects are popular and thus are demanded. Therefore, clients propagate a smaller number of CAR messages, because they require less CA information. For push-based BFR, Fig. 4.3 shows the required communication overhead for propagating content advertisements in terms of *content advertisement refresh rate* (f_r), i.e., the frequency that servers refresh CA messages.

From Fig. 4.3, we observe that for push-based BFR, the communication overhead required for propagating content advertisements increases by increasing f_r. When we compare Figs. 4.2 and 4.3, we observe that pull-based BFR requires significantly less communication overhead for propagating content advertisements compared to push-based BFR. For example, in Fig. 4.3, push-based BFR requires the least communication overhead for propagating content advertisements if $f_r = 0.017Hz$, however, even in this case, push-based BFR requires significantly more communication overhead for propagating content advertisements compared to pull-based BFR except when $\frac{1}{\delta} = 10kHz$ and α is in the $[0.6, 0.8]$ interval. Note that when pull-based BFR is used

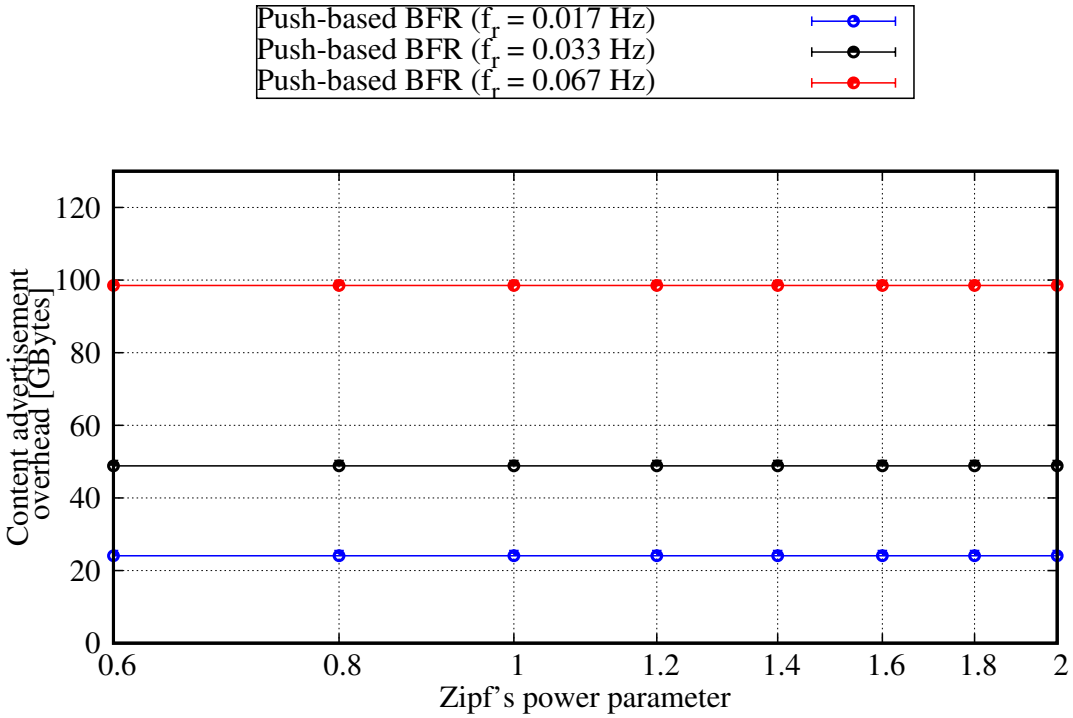


Figure 4.3: Results for content advertisement overhead for different values of α

and $\frac{1}{\delta} = 10kHz$, nodes perform very little aggregation, which is not in our interest. Therefore, in the rest of results, for pull-based BFR, we use $\frac{1}{\delta} = 2.5kHz$ and for push-based BFR, we use $f_r = 0.017Hz$. For push-based BFR, we use $\frac{1}{f_r}$ as the lifetime of CA messages. For pull-based BFR, we set the lifetime of CAR and CA messages to $4secs$ and $10secs$, respectively.

4.3.3 Storage Space Requirements for Storing Routing Information

Routing information for push-based BFR consists of CA messages, while for pull-based BFR, routing information includes both CA and CAR messages. Fig. 4.4 compares pull-based and push-based BFR in terms of average storage space a node requires to store routing information per second. For push-based BFR, we observe from Fig. 4.4 that the storage space requirements for storing routing information significantly increases with the size of the Content Universe (CU). As explained before, the reason is that using push-based BFR, clients and routers require to store the routing information for the entire CU. However, using pull-based BFR, the nodes only store the routing information for the demanded file names. Therefore, from Fig. 4.4 we observe that

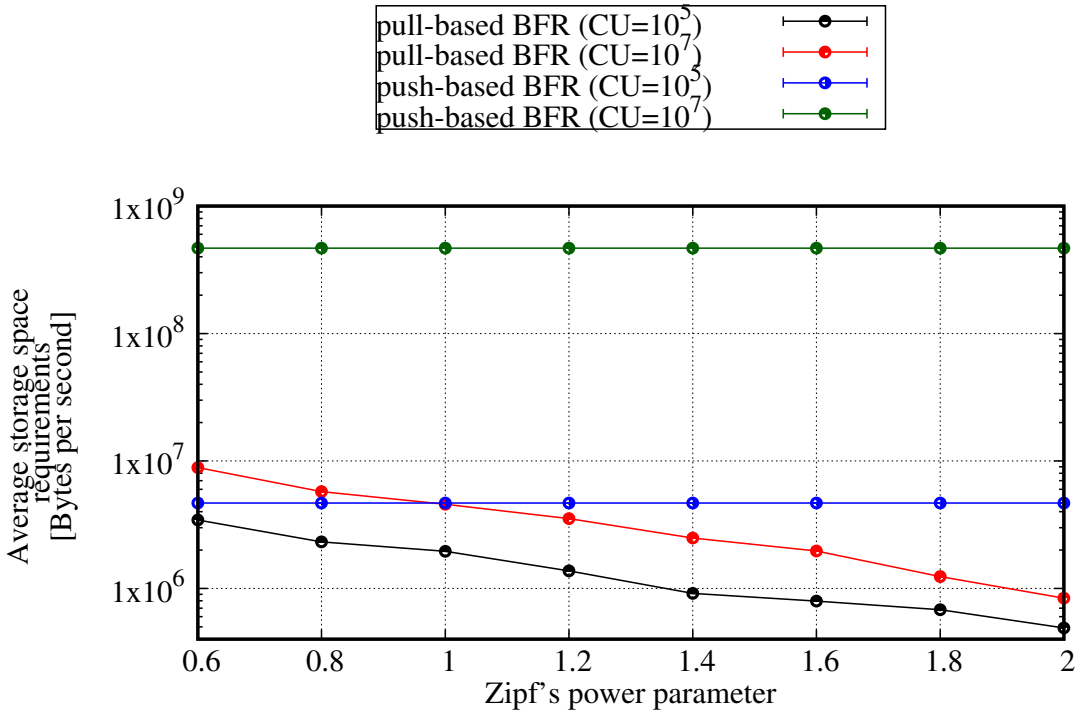


Figure 4.4: Results for storage space requirements for storing routing information for different values of α

the storage space requirements for pull-based BFR slightly grows when we increase CU size from 10^5 to 10^7. Fig. 4.4 also shows that the storage space requirements for pull-based BFR are controlled by the value of α, meaning that for higher α values, pull-based BFR has less storage space requirements, while the storage space requirements for push-based BFR only depends on the CU size. Fig. 4.4 shows that for both $CU = 10^5$ and $CU = 10^7$, pull-based BFR requires significantly less storage space for storing routing information compared to push-based BFR. Nevertheless, we observe from Fig. 4.4 that when the CU size grows from 10^5 to 10^7, pull-based BFR outperforms push-based BFR more significantly. Fig. 4.5 compares pull-based and push-based BFR in terms of the average storage space that a node needs to store routing information for one file name per second. We observe from Fig. 4.5 that pull-based BFR outperforms push-based BFR for both $CU = 10^5$ and $CU = 10^7$. If the CU size grows from 10^5 to 10^7, Fig. 4.5 shows that pull-based BFR outperforms push-based BFR more significantly.

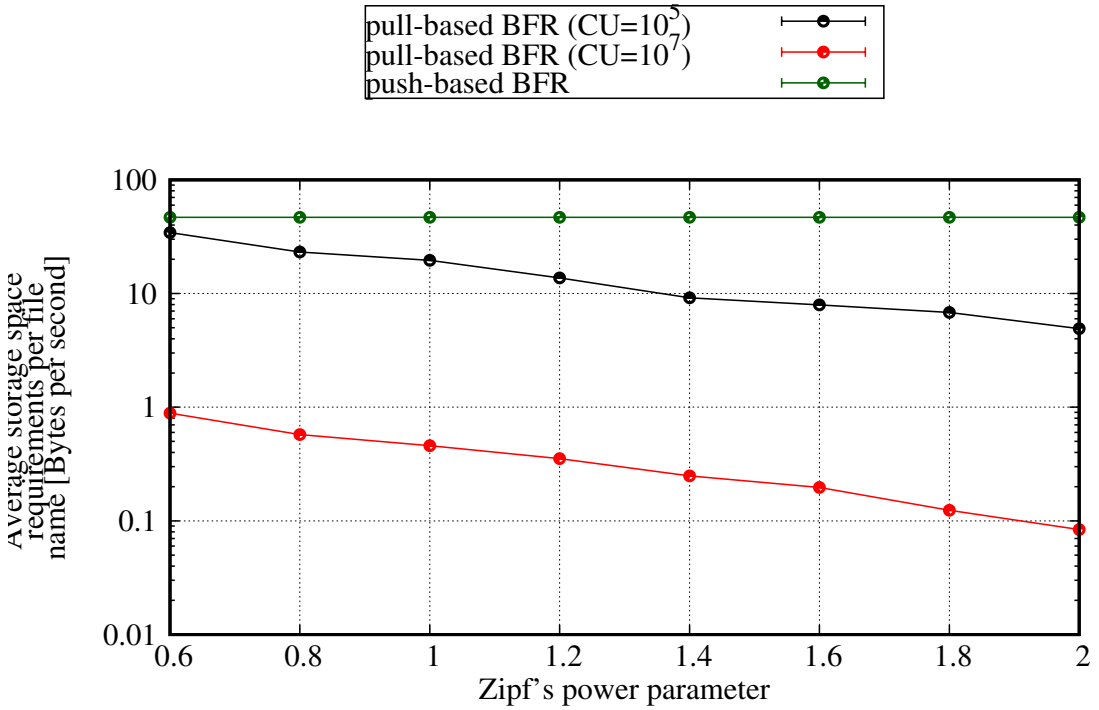
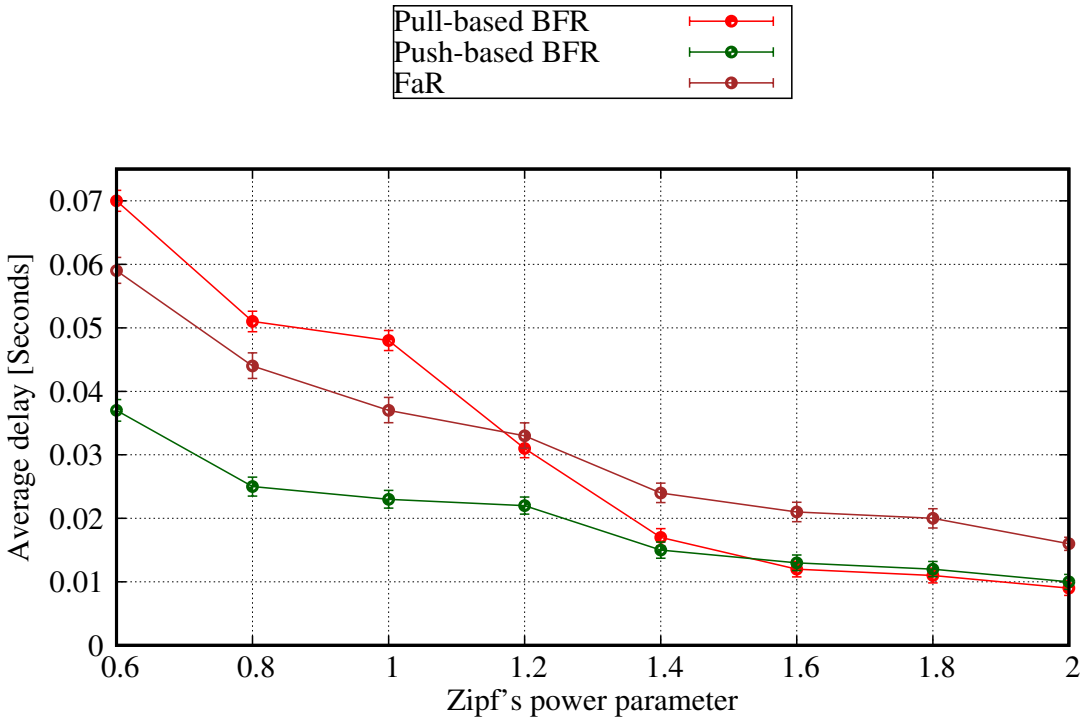


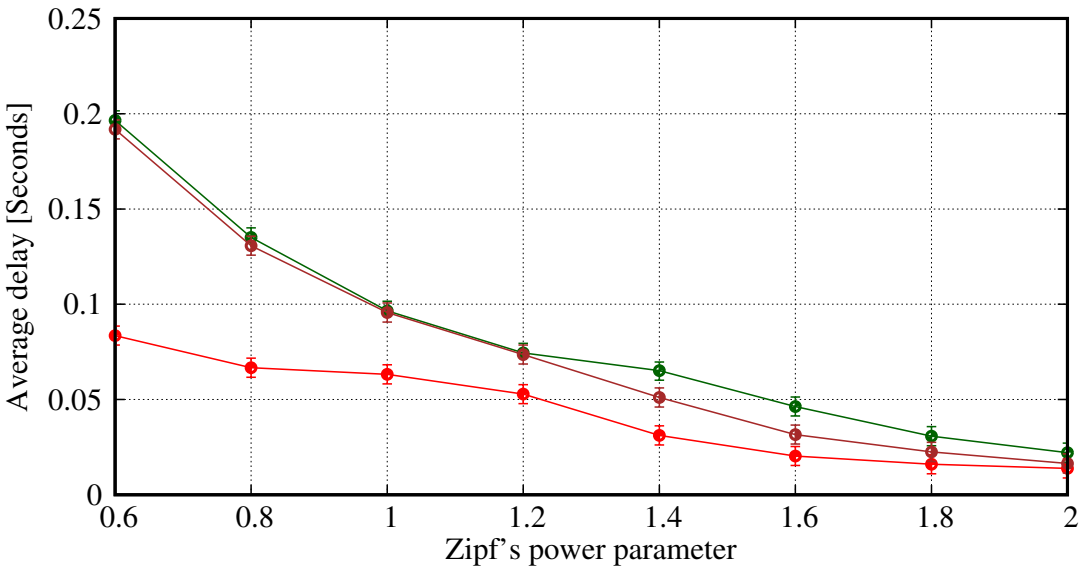
Figure 4.5: Results for storage space requirements for storing routing information per file name for different values of α

4.3.4 Average Round-trip Delay

Figs. 4.6a and 4.6b present the results in terms of average round-trip delay, i.e., the average delay that a client experiences from the time it issues an Interest to the time it retrieves the demanded Data packet. We measure this delay for all the studied protocols in two scenarios: 1) when links have full capacity, and 2) when links have only 20% of the original capacity. When users can make use of the full network capacity, Fig. 4.6a shows that if α is in the $[0.6, 1]$ interval, push-based BFR performs slightly better than pull-based BFR because the cardinality of the set of popular content objects is bigger for smaller values of α. Thus, pulling content advertisement and CAR aggregation at routers have more impact on the average round-trip delay for pull-based BFR. Nevertheless, when α is in the $[1.2, 2]$ interval, pull-based BFR and push-based BFR perform very closely to each other, because much less content objects are popular. Thus, clients need to pull much less CA information and each demanded content object will be cached close to the client that demanded it after its first retrieval. Therefore, in this case, the delay caused by pulling content advertisements and aggregating CARs and CAs has less impact on



(a) Average round-trip delay when links have maximum capacities



(b) Average round-trip delay when links have 20% of their maximum capacities

Figure 4.6: Results for average round-trip delay

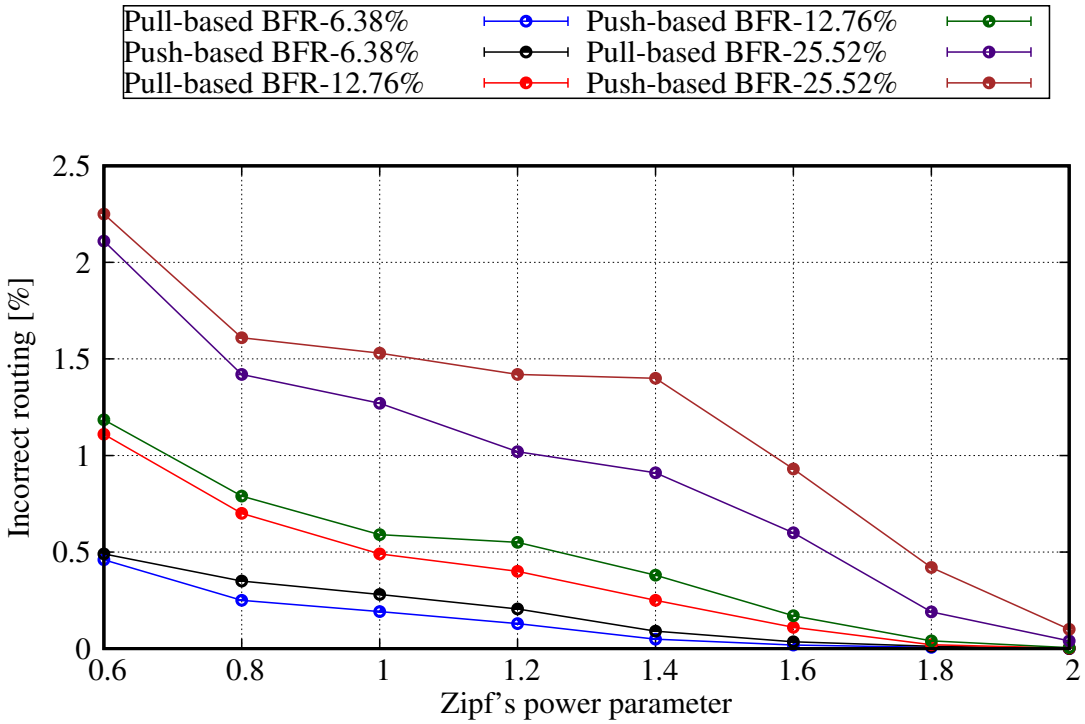


Figure 4.7: Perormance for different values of α in terms of the impact of false positive reports on routing

the overall average round-trip delay. When the links of the GEANT topology [2] have 100% of their capacities and α is in the $[0.6,1]$ interval, FaR [68] performs close to pull-based BFR. However, when α is in the $[1.2,2]$ interval, pull-based BFR outperforms FaR. We observe from Fig. 4.6b that by reducing link capacity, push-based BFR and FaR protocols are more affected, while we observe the smallest impact of limited link capacity on the performance of pull-based BFR. The reason is that pull-based BFR aggregates CAR and CA messages and, therefore, it has a much less number of transmissions than push-based BFR and FaR.

4.3.5 Impact of False Positive Errors on Routing

We analyze the impact of false positive reports on the performance of pull-based and push-based BFR. Fig. 4.7 compares these protocols in terms of the impact of false positive reports on routing. Using (2.1), we conducted experiments with $n = 1000$ and three different rates for p from set $F = \{6.38\%, 12.76\%, 25.52\%\}$ to observe the impact of false positive reports on the operation of our considered routing protocols. Fig. 4.7 shows the percentage of Interest messages that have reached wrong servers due

to false positive reports from the BFs of CA messages at routers and clients. From Fig. 4.7, we understand that the higher the value of p is, the higher the percentage of incorrect routings is, for both pull-based and push-based BFR. The reason is that when we increase the value of p for BFs, the probability that a false positive error occurs in practice is higher. Fig. 4.7 shows that the highest percentage of incorrect routing corresponds to $p = 25.52\%$. However, even in this case, only 2.25% of Interest messages have been routed towards the wrong server(s). In practice, one will not use $p = 25.52\%$ because it results in a high risk of false positive reports. Fig. 4.7 makes clear that false positive reports have less impact on the operation of pull-based BFR compared to push-based BFR. The reason is that push-based BFR stores CA messages for the entire content universe. Hence, push-based BFR stores more CA messages compared to pull-based BFR, thus, more number of BFs have to be checked. Further, from Fig. 4.7, we observe that with higher α values, false positive reports have a smaller impact on the performance of both pull-based and push-based BFR. This is due to the fact that if the value of α is higher, a smaller number of names are popular and, therefore, a smaller number of names are checked against BFs, which results in less number of false positive reports.

4.4 Conclusions

In this Chapter, we proposed pull-based BFR as a new routing protocol for NDN. Pull-based BFR has the following advantages compared to push-based BFR: 1) significantly less communication overhead for propagating content advertisements, 2) BF-based aggregation mechanism for CAR and CA messages, 3) better average round-trip delay when α is in the $[1.2, 2]$ interval, 4) less storage space requirements for clients and routers to store content advertisements, and 5) more robustness to false positive reports from BFs. Similarly to push-based BFR, pull-based BFR is fully distributed, topology agnostic, content-oriented, and does not need any IP-based routing protocol as a fall-back or primary routing mechanism. In the next Chapter, we benefit from pull-based BFR for content discovery, and we propose a network coding-based protocol to accelerate content retrieval.

5

Network Coding-based Content Retrieval based on Bloom Filter-based Content Discovery

5.1 Introduction

In Chapter 4, we described pull-based BFR and showed that it requires significantly less storage and bandwidth resources for content advertisements than push-based BFR. In this Chapter, we investigate RQ 3 of the thesis Introduction that requires to reduce content retrieval delay. Push-based and pull-based BFR forward Interest messages over multiple paths towards origin servers. Thus, there are potentially several reverse paths over which corresponding Data messages might be transmitted. As Data messages are forwarded over multiple paths, Network Coding (NC)-based forwarding can be used to maximize throughput and to reduce content retrieval delay [25]. Therefore, in this Chapter, we use our pull-based BFR for content discovery and we propose an NC-based protocol to reduce content retrieval delay. In Chapter

2, we mentioned that the works in [20, 65, 66] proposed single-session NC-based protocols for content retrieval in NDN. Differently, in this Chapter, we propose a multi-session NC-based protocol for content retrieval in NDN.

Applying multi-session NC in NDN is not trivial as we need to control the multi-session codeblock size. This has not been addressed in [20, 65, 66]. In this Chapter, we propose an NC-based protocol to address this problem. Our NC-based protocol leverages BF-based pull-based content discovery to build a distributed NC protocol based on Random Linear Network Coding (RLNC) [33] achieving a multipath Data message diffusion protocol. To manage the multi-session codeblock size, i.e., the number of linearly combined variables, the proposed NC protocol uses a maximal capacity constraint and a local feedback mechanism. To assess the performance of the proposed NC protocol, we compare it with our push-based BFR and pull-based BFR protocols that we described in Chapters 3 and 4, respectively. From the results, we observe that the proposed cooperative NC protocol achieves lower content block retrieval delay than the schemes under comparison. Further, the results reveal that the proposed NC protocol requires less bandwidth resources for content discovery than push-based and pull-based BFR.

The rest of the Chapter is structured as follows. Section 5.2 presents the proposed NC model as well as some preliminaries and definitions. Next, we describe our BF-based content discovery mechanism in Section 5.3. Then, Section 5.4 presents the proposed network code selection and content forwarding protocol. After, we discuss the received Data message processing method in Section 5.5. Finally, Section 5.6 discusses performance evaluation and Section 5.7 concludes the chapter.

5.2 Network Coding Model

Let V be the set of nodes in the network with $S \cup U \cup R = V$, where S is the set of source nodes (servers), U is the set of clients, and R is the set of routers. Each node $v \in V$ has at time t a set of neighbors $N_t(v)$. Finally, let x_i^j be the i^{th} original Data segments generated at source s_j, $j = 1, ..., m$.

A network coded Data message contains the identities of n variables $P_i, i \in [1, n]$ as well as the coding coefficients. If a variable is not yet decoded at a node, we call it a *decoding* variable for that node. For simplicity, we assume that node v divides its CS memory space into two buffers: a decoded buffer $\mathscr{B}^v$ containing all Data messages

Face 1	Face 2	Face 3
IBF	IBF	IBF
decodingBF	decodingBF	decodingBF
decodedBF	decodedBF	decodedBF
Available capacity	Available capacity	Available capacity

Figure 5.1: Neighborhood state array for router R_4.

that have been decoded by this node and a decoding buffer $\mathscr{A}^v$ containing m^v linear combinations of decoding variables. We use in the following the notation $x_i \in \mathscr{A}^v$ to indicate that the decoding variable x_i is used in a linear combination stored in the decoding buffer. We consider that a node v can accept only up to C^v decoding variables in its equation system, with C^v forming the capacity constraint. Therefore, node v discards a received Data message that results in having more than C^v decoding variables in its equation system. In Section 5.4, we discuss the important operational purpose of this capacity constraint for multi-session codeblock size management. Our NC-based content forwarding protocol requires each node to maintain the state of all its neighbors consisting of the sets of decoded and decoding variables. Each node updates its neighbors by transmitting information about its state at both BF-based content discovery and BF-based local feedback Interest transmissions phases, which we describe in the following. Fig. 5.1 shows the structure of neighborhood state array for router R_4 in Fig. 5.2. In the following, we explain how nodes receive and update their neighbor state information.

5.3 Bloom Filter-based Content Discovery

Our BF-based content discovery is inspired by the BF-based pulling of the required content advertisements we used in pull-based BFR protocol [51]. The main difference is that pull-based BFR transmits BF-based aggregated Interest messages to pull the required content advertisements, while in this Chapter, clients send BF-based aggregated Interest messages to pull network coded Data messages.

To explain the BF-based content discovery protocol, we assume that in Fig. 5.2 clients C_X and C_Y create Interests $I(N_1)$ and $I(N_2)$ to request two Data messages with names N_1 and N_2, respectively. Clients C_X and C_Y store Interests $I(N_1)$ and $I(N_2)$ in their PITs. Nonetheless, client C_X keeps $I(N_1)$ pending and creates a BF called Interest Bloom Filter (IBF) containing the hashed value of N_1 and stores this IBF in an Aggregated

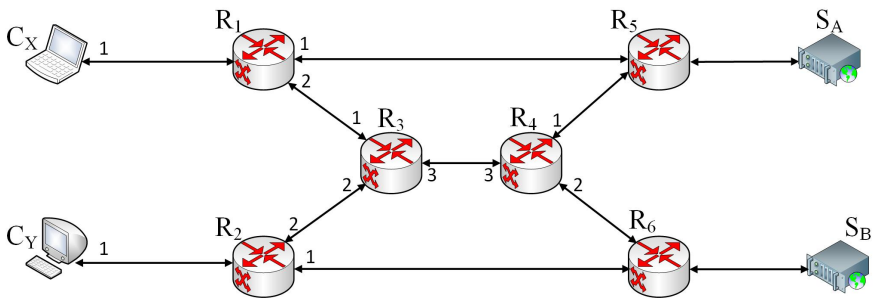


Figure 5.2: A topology for describing our NC-based protocol.

Interest Message (AIM) called AIM_{C_X} with name prefix $/AIM/C_X$. Client C_Y follows the same procedure and keeps $I(N_2)$ pending creating an IBF containing the hashed value of N_2 stored into another AIM called AIM_{C_Y} with name prefix $/AIM/C_Y$. Fig. 5.3 shows the AIM structure. Note that all the nodes use a hash function with the same seed for calculating the hashed values of names. Then, client C_X sends AIM_{C_X} to router R_1 and client C_Y sends AIM_{C_Y} to router R_2. In general, we assume that each client has only one outgoing face to the only router connected to it. When a router receives an AIM, it waits for a short time interval t_{aggr} to collect other AIMs sent by other nodes (t_{aggr} is of msecs scale). After waiting t_{aggr}, router R_1 does not receive any other AIMs from other clients. Thus, R_1 sends AIM_{C_X} over faces 1 and 2 to R_5 and R_3. The same procedure takes place at R_2 and this router sends AIM_{C_Y} to R_3 and R_6. We assume that R_3 receives AIM_{C_X} and AIM_{C_Y} within a time interval $t \leq t_{aggr}$ so that R_3 can aggregate AIM_{C_X} and AIM_{C_Y} and sends the aggregation of these messages called AIM_{R_3} with name prefix $/AIM/R_3$ over face 3. To aggregate AIM_{C_X} and AIM_{C_Y}, router R_3 makes a union of the IBFs of these messages. At the same time, R_3 sends AIM_{C_X} over face 2 and sends AIM_{C_Y} over face 1. In general, if a router has received AIMs over a set of faces $F = \{f_1, f_2, ..., f_k\}$ and IBF_{f_1} is the IBF calculated for an AIM that will be sent over face $f_1 \in F$, the IBF of face f_1 is calculated from (5.1).

$$IBF_{f_1} = \{ \bigcup_{\forall k \in F} | k \neq f_1 \} \tag{5.1}$$

After IBF aggregation, R_3 sends AIM_{R_3} over face 3. Note that R_3 adds face 3 to the out-records of both messages AIM_{C_X} and AIM_{C_Y} to record that both messages have been sent over face 3. For future aggregations, R_3 will not use the AIMs that it had issued itself. It can figure this out by checking the second name component of the AIMs. Though we discussed only R_3, all other routers pursue the same aggregation and forwarding strategies for the received AIMs.

/AIM/Cx/0x00
Nonce
Lifetime
Available capacity
IBF bit vector and hashing function information
decodingBF bit vector and hashing function information

Figure 5.3: AIM structure.

To have practical IBF aggregation operations, we assume that all the nodes create IBFs with equal sizes. In Eq. (2.1), if we specify the values of m and p, we will derive the maximum value for n. The maximum value of n is the maximum number of elements that could be inserted into the IBF. Note that in distributed systems, we can have such rules that nodes use the same word, "ICN", for example, to generate the hash functions or using the same value for m. This is in analogy to what happens in IP networks where the IP address size is the same constant value for all the nodes. It is important to emphasize that a node can make a union of two IBFs provided that the number of inserted elements into the resulting IBF does not exceed its capacity. In Chapter 4, we proposed practical BF aggregation strategies to this aim, which we use also in this Chapter.

For content discovery, clients issue AIMs and routers aggregate them on their paths towards the servers. However, besides this content discovery method, we assume that each node frequently sends local feedback information to its neighbors about its state, i.e., the identities of the variables stored in the decoding and the decoded buffers, as well as the remaining capacity for accepting new variables in the linear combinations (packets) that will be received in the future. Besides updating the neighborhood about state information, a node sends local feedback transmissions to enquire neighbors whether they have information that can help it to decode its current decoding variables faster by returning useful linear combinations. Let us explain this process for router R_4 using Fig. 5.2. In this figure, we assume that router R_4 has two sets at any time instant t, namely: 1) the set of decoding variables for node R_4, called $\mathscr{R}^{R_4,t}$ characterized by the BF $\Phi_R^{R_4,t}(.)$, which contains all variables that exist in linear combinations stored by node R_4 up to time t; and 2) the set of decoded variables $D^{R_4,t}$ characterized by the BF $\Phi_D^{R_4,t}(.)$, which contains all variables that node R_4 has decoded up to time t. $\Phi_D^{R_4,t}(.)$ is a big BF obtained over the set of last W decoded variables that are chosen in the order of 200 variables and $\Phi_R^{R_4,t}(.)$ is

/FIM/R3/0x00
Nonce
Lifetime
Available capacity
decodedBF bit vector and hashing function information
decodingBF bit vector and hashing function information

Figure 5.4: FIM structure.

a small BF with a size of about 20 that is the maximum capacity constraint for each router. These two BFs are updated every time new variables are decoded by removing the oldest decoded variables to allow the insertion of new variables. Besides these BFs, node R_4 has an available capacity $c^{R_4}(t) \leq C^{R_4}$ for accepting new variables that node R_4 neither has decoded nor has stored them as decoding variables up to time t.

When node R_4 wants to transmit a local feedback, it creates an Interest message called Feedback Interest Message (FIM) that carries $c^{R_4}(t)$ value as well as both BFs $\Phi_R^{R_4,t}(.)$ and $\Phi_D^{R_4,t}(.)$. Fig. 5.4 shows the FIM structure. When router R_4 creates this FIM, it sends it over each face $i (i = 1,2,3)$ with name prefix $/FIM/R4/f_i$. The second name component of the FIM corresponds to the sender node ID (R_4) and the third name component indicates the face ID over which the FIM is sent. Note that each node sends FIMs frequently (e.g., every 1 sec) to announce its state to the neighborhood and FIMs do not travel more than one hop. When a node receives an FIM, it selects a network code according to the information signaled by the received FIM and it encapsulates the calculated network code into a Data message called Feedback Data Message (FDM) that is returned in response to the received FIM. We describe the network code design algorithm in Section 5.4.

Any node v that wants to forward an AIM, stores its BF $\Phi_R^{v,t}(.)$ into the sent AIMs because by sending $\Phi_R^{v,t}(.)$ with AIMs, node v signals its decoding variables more frequently. As mentioned earlier, $\Phi_R^{v,t}(.)$ is a small BF, which requires small bandwidth resources for transmission.

If at time instant t node v receives from neighbor n_i an FIM or an AIM over face f_j, node v updates the feedback information, received in the FIMs or AIMs, for face f_j in the neighborhood array state.

5.4 Network Code Selection for Content Forwarding

Network code selection, i.e., choosing the variables for a linear combination, takes place when: 1) a source node receives an AIM and has to reply with a network coded Data message, or 2) a node receives an FIM and has to reply with a network coded FDM. We develop a model to calculate the *utility* of each variable to be used in network codes. When a node calculates the utilities for its stored variables, the node solves an instance of a linear program to calculate the probabilities of selecting variables for the linear combination to forward so that the objective of sending the maximum requested information is reached. Therefore, the content forwarding algorithm is responsible to decide the variables to combine among all variables stored in decoding and decoded buffers. We assume that the nodes are non-selfish and behave cooperatively. When a node v receives an AIM/FIM over a face f, node v aims to combine the maximum amount of the information that has been requested over face f into the returned network coded Data message/FDM. As previously mentioned, the capacity constraints of neighbors should not be violated. To this end, each node uses the neighbors' states gathered inside an array and updated by node feedback piggybacked in received AIMs and FIMs.

With this goal in mind, let us define for each variable x_i in node v (regardless of the fact that it is in decoded or decoding buffer) an expected utility $u^{iw}(t)$ for combining this variable into a linear combination that is going to be returned to the neighbor w. Based on the feedback information received from neighbor w and stored in the neighborhood array, we calculate the $u^{iw}(t)$ values as follows.

- If variable x_i did not exist in the last IBF received from neighbor w, then it is considered as an unsolicited variable. Therefore, the utility of combining variable x_i in the Data message/FDM returned to neighbor w is zero, i.e., $u^{iw}(t) = 0$.

- If variable x_i has been decoded by node w ($i \in \mathscr{B}^w(t)$), the utility of combining this variable in the Data message returned to node w is zero and $u^{iw}(t) = 0$.

- If variable x_i has not been received by node w ($i \notin \mathscr{B}^w(t) \cup \mathscr{A}^w(t)$) and node w has still capacity to accept new variables, combining this message will result in spreading a new solicited variable to node w, and there is a non-zero utility in combining this variable. Therefore, the expected benefit of combining the variable x_i to forward to node w is set as $u^{iw}(t) = 1$.

- If the variable x_i is a decoding variable for node w ($i \in \mathscr{A}^w(t)$), the utility of combining this variable depends on the number of places in the decoding buffer that sending this variable can free because of the variables are retrieved by means of decoding. These free spaces might be used in the next step to spread new solicited variables, which is the main goal of a node in the network. Adding a new equation in a system of $m^w(t)$ equations can lead at best to the retrieval of $m^w(t)+1$ variables. Therefore, an optimistic view will consider a utility equal to $u^{iw}(t) = m^w(t)+1$.

In very general terms, a large spectrum of utility values might be used to account for different application-related constraints; for example, reducing decoding delay at clients, giving priority to the decoding of particular messages, node selfishness, etc. However, as explained above, we assume non-selfish nodes that are willing to assist the content retrieval process in the network.

Let us assume that the probability of choosing variable i in the returned linear combination is p_i. The global expected utility of forwarding a linear combination carried by the returned Data message/FDM from the sender node v to the receiver neighbor w at time t could be defined as:

$$\mathscr{U}^v(t) = \sum_{x_i \in \mathscr{A}^v(t) \cup \mathscr{B}^v(t)} p_i u^{iw}(t) \tag{5.2}$$

The aim of node v is to maximize this global utility subject to the set of constraints defined below:

$$\begin{aligned} \underset{p_i}{\text{maximize}} \quad & \mathscr{U}^v(t) \\ \text{subject to} \quad & c^w(t) - \sum_{x_i \notin \mathscr{B}^w(t) \cup \mathscr{A}^w(t)} p_i \geq 0, \\ & 0 \leq p_i \leq 1. \end{aligned} \tag{5.3}$$

In Eqs. (5.2)-(5.3), the parameters $u^{iw}(t)$, $\mathscr{A}^w(t)$ and $\mathscr{B}^w(t)$ could be obtained from the feedback received from neighbor w. Therefore, they are known values, independent of p_i values. Thus, the optimization (5.3) is an instance of classical linear programming that could be solved by the simplex method in polynomial time. Node v solves the linear program to calculate the p_i probabilities. Then, node v chooses the

Table 5.1: Reports from feedback BFs of node w for a variable P_i

Report from $\Phi_D^{w,t}(.)$	Report from $\Phi_R^{w,t}(.)$
Negative	Negative
Negative	Positive
Positive	Negative
Positive	Positive

variables to be combined by sampling uniformly at random following the distribution given by p_i probabilities.

When node v wants to select network codes to send a Data message/FDM to node w, let us see the effect of false positive errors for a variable x_i in BFs $\Phi_D^{w,t}(.)$ and $\Phi_R^{w,t}(.)$. Table 5.1 shows all the four states of reports that might happen at these two BFs. The first line of Table 5.1 corresponds to a state where both BFs report negative. When we use BFs, false negative errors are impossible. Therefore, both reports from BFs $\Phi_D^{w,t}(.)$ and $\Phi_R^{w,t}(.)$ are definitely correct. The second line of Table 5.1 indicates a positive report from $\Phi_R^{w,t}(.)$. If this is a false positive report, the worst case that can happen is that a new variable will be combined and considered a decoding variable for the receiver. Since receivers do not accept new variables more than their capacity, there is no problem. If the state indicated at line 3 of Table 5.1 happens and the report from $\Phi_D^{w,t}(.)$ is a false positive report, then the worst case is that the sender assumes variable x_i as a decoded variable and does not combine it in the returned Data message. If the state indicated in the fourth line of Table 5.1 happens, one of the reports from $\Phi_R^{w,t}(.)$ or $\Phi_D^{w,t}(.)$ is a false positive report. In such a case, the sender combines variable x_i in the returned Data message to prevent any negative effect from the happened false positive report.

5.5 Received Data Message Processing

Let us assume that node v has stored a set of m equations and n variables. Upon reception of a new linear combination in a Data message/FDM, the node removes from it the decoded variables that can be found in the decoded buffer. This results in a new linear combination that only contains decoding or new variables. To enforce the maximum capacity constraint, a linear combination containing k new variables will be accepted and further processed only if $n + k \leq C^v$, otherwise the received

Data message/FDM is discarded. This simple mechanism ensures that the number of variables does not increase out of control, and, therefore, manages multi-session codeblock size.

When a linear combination is accepted, it is stored in the decoding buffer. The coefficients of the stored linear combination are extracted and are placed into a decoding matrix G_v with the maxImum size of $C^v \times C^v$. A Gaussian elimination algorithm is applied to matrix G_v and to the linear combinations stored in the decoding buffer to construct a matrix in row echelon form G_v. Gaussian elimination could result in the following situations.

1. The newly received linear combination results in an all-zero line. In this case, the received packet is non-innovative and the packet is tagged as *redundant*. This packet is removed from the decoding buffer and the rank of the linear system does not change.
2. The newly received linear combination results in a line with a single non-zero value. In this situation, at least a single variable has been retrieved. Replacing this decoded variable in the previous equations and removing it from the equation system reduces the rank by one.
3. The newly received linear combination reduces the rank of the system by one, without any variable being decoded.

Whenever one or several variables are decoded, they are moved to the decoded buffer. The capacity c^v and the decoded and decoding BFs are updated accordingly.

5.6 Performance Evaluation

In this section, we evaluate the performance of the proposed NC-based protocol and compare it with that of push-based and pull-based BFR. All schemes are implemented in ndnSIM2.1 [56].

5.6.1 Simulation Settings

To compare the proposed NC-based protocol with push-based and pull-based BFR, we use the GEANT topology [2] illustrated in Fig. 3.6. Then, we randomly connect

10 servers and 50 clients as leaf nodes to it. Overall, the topology has 100 nodes. We use real HTTP request traces [26] to extract a URL dataset and assume that content popularity follows a Zipf-Mandelbrot distribution (we showed the Zipf-Mandelbrot probability distribution formula in Eq. (3.2)) where we use α values from the [0.6,2] interval. The content universe consists of 100,000 content objects, each is divided into 100 segments. In Eq. (2.1), we use $n = 100$ for IBFs, $n = 20$ for decodingBFs, and $n = 200$ for decodedBFs, respectively. We use $p = 0.0638$ as the false positive probability of the BFs. The presented results are averaged over ten simulations and the reported mean values have 95% confidence intervals. In the following, we analyze the performance of the proposed NC protocol, the push-based BFR, and the pull-based BFR based on two metrics: 1) content discovery overhead, and 2) average content block retrieval delay.

5.6.2 Content Discovery Overhead

In Fig. 5.5, we show results in terms of the bandwidth used for content discovery with different values of Zipf parameter α. Fig. 5.5 makes clear that as α value increases, content discovery requires less bandwidth for both the proposed NC protocol and pull-based BFR. This behavior is expected for pull-based BFR because when α value is larger, less content objects are popular, and, therefore, clients need to pull content advertisements for less content object names. With the proposed NC protocol, when less content objects are popular, they will be decoded faster at caches that are close to the clients, which results in smaller content discovery overhead. Fig. 5.5 makes clear that for all α values, the proposed NC protocol uses much less bandwidth for content discovery than for pull-based BFR. The reason for this is that the proposed NC protocol excludes the content advertisement phase.

In Fig. 5.6, we show the content discovery overhead for push-based BFR with different content advertisement refresh frequencies. From Fig. 5.6, we can see that push-based BFR requires significantly higher bandwidth resources for content discovery than both pull-based BFR and the proposed NC protocol. This happens because push-based BFR needs to push the BF-based content advertisements for all the provided content object names.

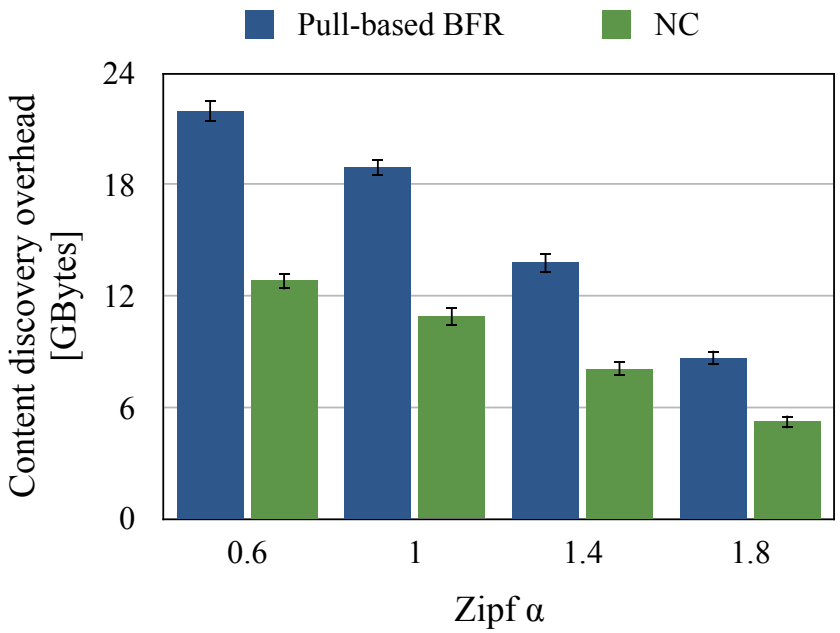


Figure 5.5: Content discovery overhead with different α values for our NC protocol and pull-based BFR.

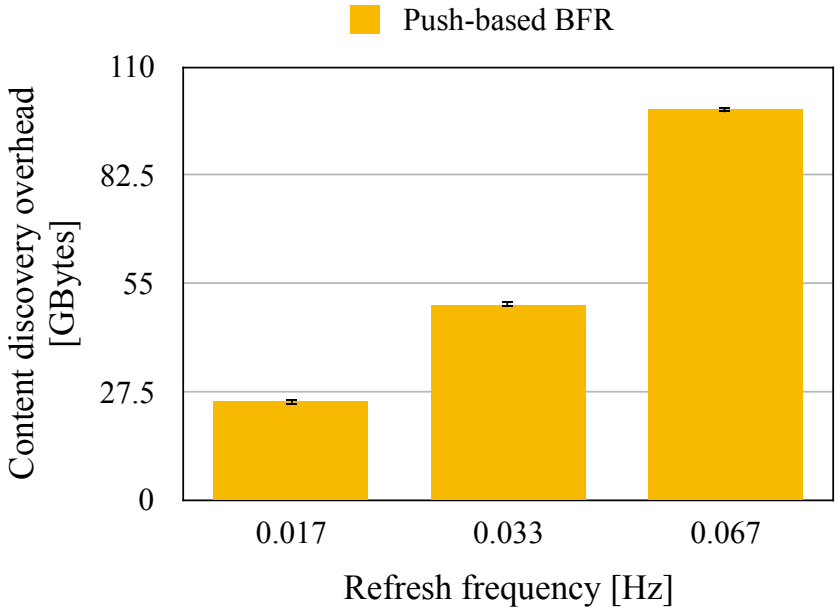


Figure 5.6: Content discovery overhead for push-based BFR.

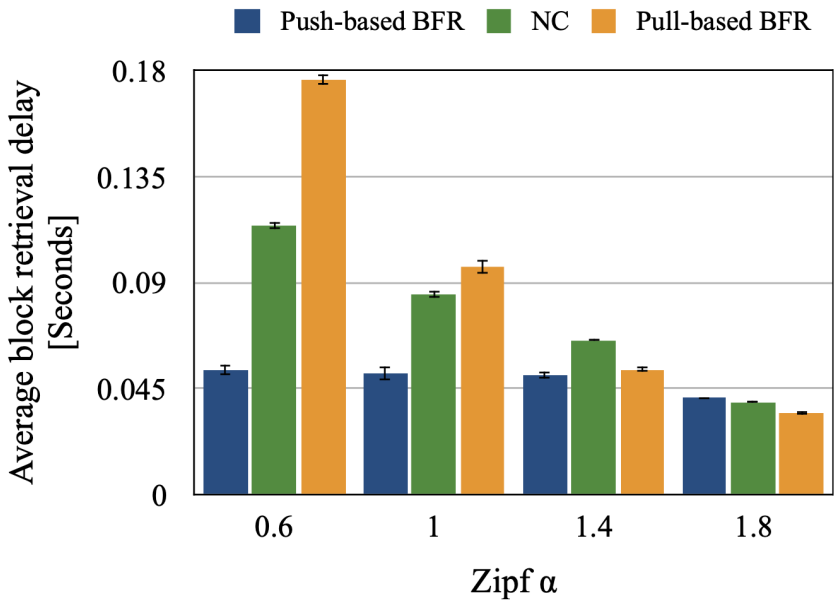


Figure 5.7: Content block retrieval delay for the proposed NC protocol with full link capacities.

5.6.3 Average Content Block Retrieval Delay

We evaluate the performance of all the considered protocols in terms of average content block retrieval delay, i.e., the average delay from the time clients send AIMs until the time they retrieve a block of segments of the requested content objects. We consider a block size of 20 segments, which is equal to the maximal capacity constraint of the nodes. Although the GEANT testbed is a high-speed network, we calculate average content block delays in four scenarios: 1) links have full capacities, 2) links have 50% of their original capacities, 3) links have 20% of their original capacities, and 4) links have 10% of their original capacities. This evaluation strategy helps us observe the performance of all protocols in scenarios with ample and restricted link capacities.

From Figs. 5.7 and 5.8, we show the delay for 100% and 50% link capacities and we note that, the average content block retrieval delay decreases by increasing α value. This is expected because, when α value is larger, fewer content objects are popular, which will be soon cached (in non-NC scenarios) or decoded (in NC scenarios) closer to the clients. Further, Figs. 5.7 and 5.8 make clear that when α is in [0.6,1], pull-based BFR has much higher delay than push-based BFR and the proposed NC protocol. The reason for this is that more content objects are popular for which pull-based BFR

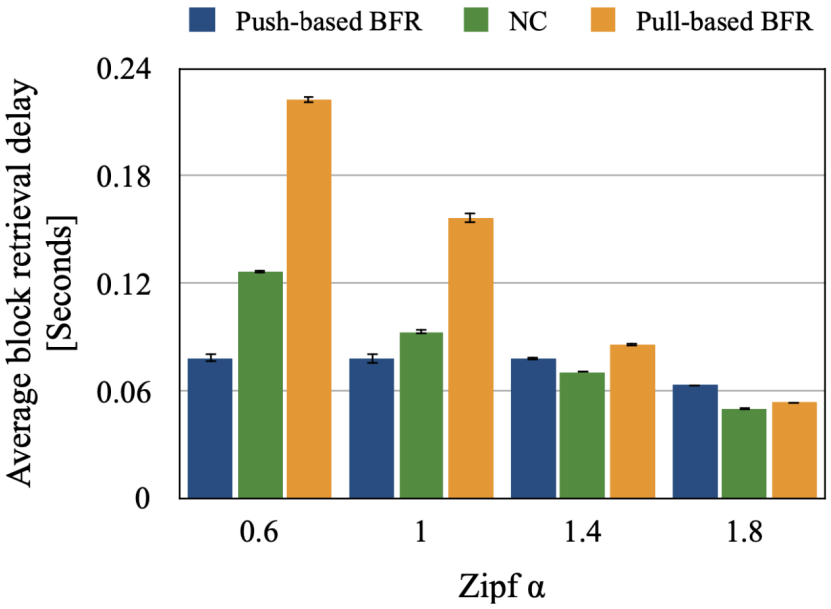


Figure 5.8: Content block retrieval delay for the proposed NC protocol with 50% of link capacities.

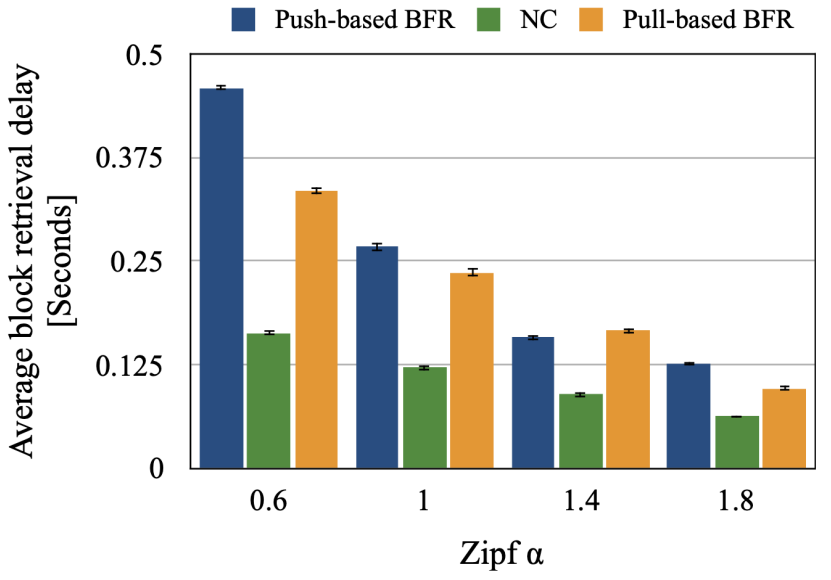


Figure 5.9: Content block retrieval delay for the proposed NC protocol with 20% of link capacities.

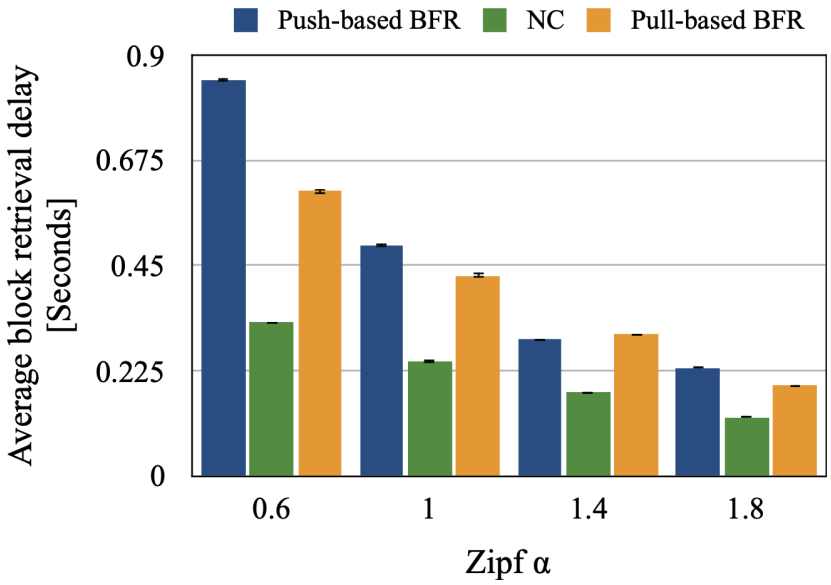


Figure 5.10: Content block retrieval delay for the proposed NC protocol with 10% of link capacities.

requires to pull the content advertisements that entails delay. When the α values are in [1.4, 1.8], we see from Figs. 5.7 and 5.8 that pull-based BFR and the proposed NC protocol achieve similar results due to the fact that mostly the requested content objects are cached/decoded closer to the clients. When we reduce the link capacities from 100% to 50% (Fig. 5.8), we note that, the performance of push-based BFR starts to degrade. On the other hand, the performance of pull-based BFR and the proposed NC protocol start to improve because they do not need to push the content advertisements frequently. Figs. 5.9 and 5.10 present delay comparison when all the links have 20% and 10% of their original capacities. We observe from these figures that the proposed NC protocol significantly outperforms push-based BFR and pull-based BFR. The results demonstrate that when link capacities are limited, NC-based communications offer significant gains.

5.7 Conclusions

In this Chapter, we presented a cooperative NC-based content retrieval based on BF-based content discovery for NDN. For content discovery, we used a BF-based protocol similar to pull-based BFR that we described in Chapter 4. To retrieve content objects, the proposed cooperative NC protocol uses the information received in AIMs and FIMs to calculate the utility of variables and to design network codes. We evaluated

the performance of our NC protocol and compared it with that of push-based and pull-based BFR, that we described in Chapters 3 and 4, respectively. The proposed cooperative NC protocol requires significantly less bandwidth resources for content discovery than both push-based and pull-based BFR. Further, the proposed protocol achieves lower content block retrieval delay compared to that of push-based and pull-based BFR. In this Chapter and the previous two Chapters, we proposed routing and content retrieval protocols for NDN. The next Chapter investigates routing in SCN and proposes clustering algorithms and BF-based routing protocols for L-SCN [31].

6

Bloom Filter-based Routing for Dominating Set-based Service-Centric Networks

6.1 Introduction

In Chapters 3 and 4, we presented BF-based routing protocols for NDN. In this Chapter, we investigate routing in SCN. From the Introduction, we recall that, L-SCN presents a layered routing architecture for SCN, where a so-called *supernode (SN)* is responsible for managing its domain as well as for communicating with the SNs of other domains to perform inter-domain routing. In the Introduction, we posed RQ 4 related to SN selection for L-SCN. In this Chapter, we use Dominating Sets (DS) and Connected Dominating Sets (CDS) [44] to select appropriate nodes as SNs in the network topology. We propose fully distributed algorithms for constructing DS as well as CDS over the network topology. When SNs are selected and nodes are clustered, the nodes of each domain inform their SNs about their available service names and resources (e.g., CPU,

RAM) to prepare routing information. To this aim, the nodes use BF which reduces bandwidth and storage overhead.

We assess the performance of the proposed routing protocols over various real topologies [2, 7] and observe that: 1) the bandwidth overhead required to construct DS and CDS increases with the topology size, 2) the proposed CDS construction algorithm requires more bandwidth overhead than our DS construction algorithm, 3) for large topologies, the proposed CDS-based routing protocol requires drastically less bandwidth overhead to route service requests than both our DS-based routing protocol and vanilla NDN with multicast strategy [89], and 4) we observe that the CDS-based routing protocol achieves slightly better service retrieval time than DS-based routing, while both DS-based and CDS-based routing protocols have much less service retrieval time compared to NDN with multicast strategy.

The rest of the Chapter is structured as follows. Section 6.2 discusses the proposed algorithms for constructing both DS and CDS. Section 6.3 and Section 6.4 describe routing in DS-based and CDS-based SCNs, respectively. In Section 6.5, we discuss performance evaluation of our protocols and we conclude the Chapter in section 6.6.

6.2 Clustering Network Nodes

In this Chapter, we first propose an algorithm for constructing a DS. Then, we suggest an algorithm for converting a DS to a CDS. In the following, we present our algorithms for DS as well as CDS construction to cluster the nodes in the network topology.

6.2.1 Dominating Set Construction

Our DS construction algorithm enables the nodes to store the following parameters:

- **snFlag:** a flag that indicates whether a node is an SN or not.

- **snFaceId:** the face ID of the face over which the SN of the associated domain can be reached.

To describe our algorithm, we use a part of the GEANT topology [2] illustrated in Fig. 6.1. In this topology, nodes represent core routers located in European countries. In

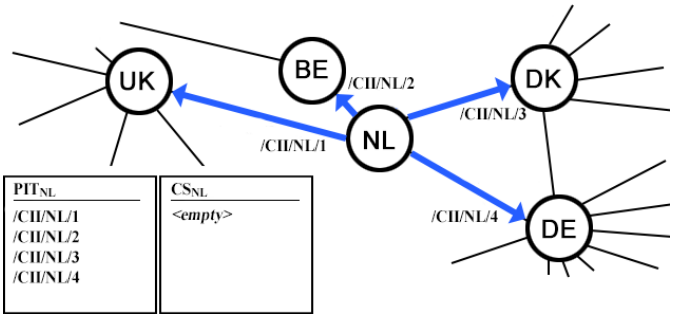
Fig. 6.1, we focus on the node with label *NL* that represents the Netherlands and shows the message exchanges needed for constructing a DS. Interest messages are shown with blue arrows and Data messages with green arrows. We only show the messages sent by *NL*, but the algorithm runs in parallel for each node.

1. The algorithm starts with node *NL* sending a Clustering Initializing Interest (CII) message with name $/CII/NL/faceID$ over each of its faces to ask each of its neighbors to send back the number of its neighbors (Fig. 6.1a). The neighbors of node *NL* store the received CII messages in their PIT tables.

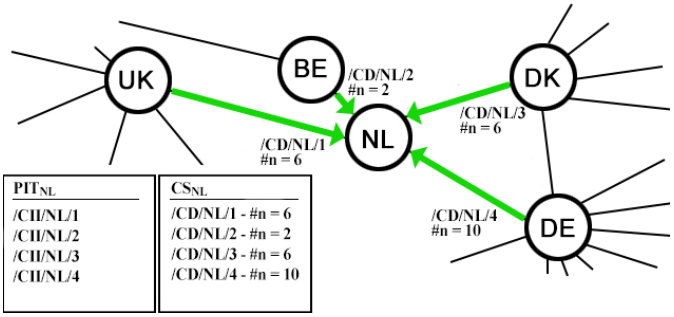
2. In Fig. 6.1b, we show that when *NL*'s neighbors receive a CII message, they respond with a Data message called Clustering Data (CD) message with the same name containing the number of its neighbors as well as its node ID. When the CD message arrives at the node that issued the corresponding CII message, it is stored in the CS table. Then, the face ID through which the CD message is received is stored in the CS entry associated with the received CD message. In such a case, the PIT entries of the CII messages are satisfied and are deleted from the PIT.

3. In Fig. 6.1b, node *NL* waits for a short time interval t_w to receive CD messages from all its neighbors. Then, node *N*L compares the number of its neighbors with that reported by its neighbors (note that if node *NL* has more neighbors than its neighbors, it appoints itself as an SN). Then, as Fig. 6.1c shows, node *NL* sends a Supernode Selection Interest (SSI) message with name $/SSI/NL$ to its neighbor with the highest number of neighbors, which is node *DE*. If a node has multiple neighbors that have the same highest number of neighbors, the node randomly selects one of them and appoints it as an SN.

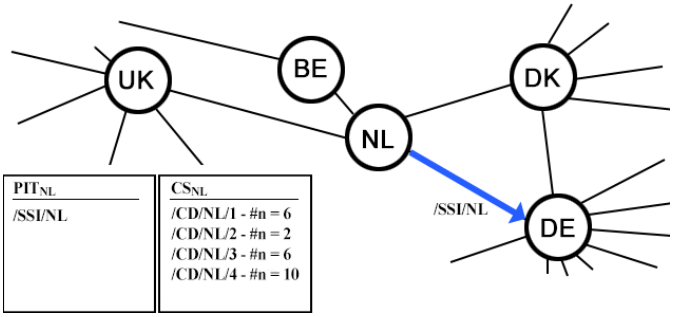
4. After node *DE* receives the SSI message from *NL* (see Fig. 6.1d), it appoints itself as an SN and responds to *NL*'s SSI message with a Supernode Selection Data (SSD) message. When node *NL* receives the SSD message, it sets its SN face ID to the face ID over which the SSD message was received from node *DE*.



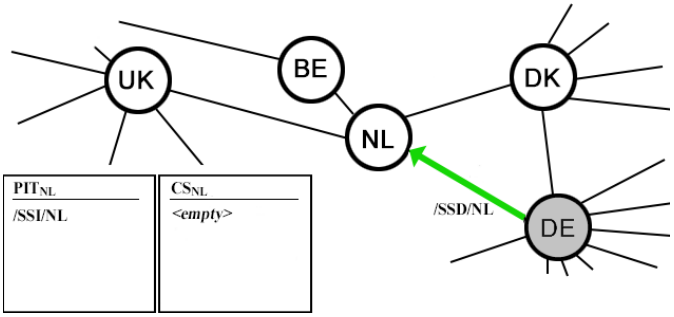
(a) Node NL sends a CII message to its neighbors



(b) Each neighbor replies with a CD message carrying the number of its neighbors



(c) Node NL sends an SSI message to node DE to select it as an SN



(d) Node DE makes its snFlag true and acknowledges node NL with an SSD message

Figure 6.1: The clustering algorithm described with a part of the GEANT network.

6.2.2 Connected Dominating Set Construction

After running our DS construction algorithm (Section 6.2.1), we obtain a DS, where every node is either a dominator, in our case an SN, or a dominated node, which we call hereafter a regular node. This property assures that the distance between any two SNs does not exceed three hops. Therefore, the maximum distance for a message from an SN to reach at least another SN is also three hops. To construct a CDS, we let every SN know its nearest SNs. Using this knowledge, each SN decides the nodes to use for connecting the DS so that it will become a CDS (the nodes in between the nearest SNs). Our protocol does not permit the messages sent for CDS construction to travel more than three hops. In addition to the parameters *snFlag* and *snFaceId* mentioned in Section 6.2.1, our CDS construction algorithm enables all SNs to also store the following parameters:

- **cFlag** is a flag that indicates whether the associated SN has already run the proposed CDS construction algorithm or not.

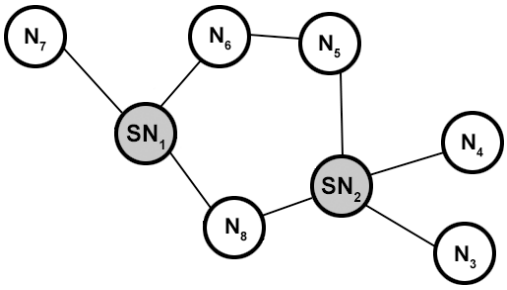
- **snFaceList** is a list containing the face ID(s) over which the associated SN can reach other SNs.

We show a small topology, in Fig. 6.2a, to describe how the proposed CDS construction algorithm connects the DS previously constructed by our DS construction algorithm. For the sake of simplicity of presentation, we only show the messages sent by SN_1, but similar operations are applied in parallel for each SN.

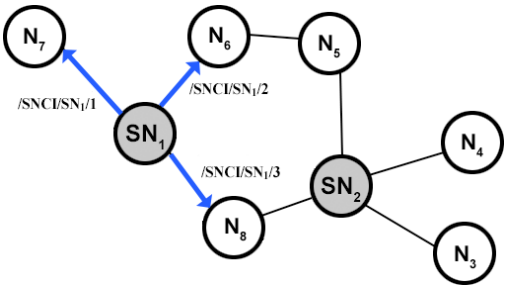
1. The algorithm starts with SN_1 sending over each face i a Supernode Connection Interest (SNCI) message with name $/SNCI/SN1/i$ to all of its neighbors, as shown in Fig. 6.2b.

2. In Fig. 6.2c, the neighbors of SN_1 proceed with forwarding the received SNCI messages until another SN is reached.

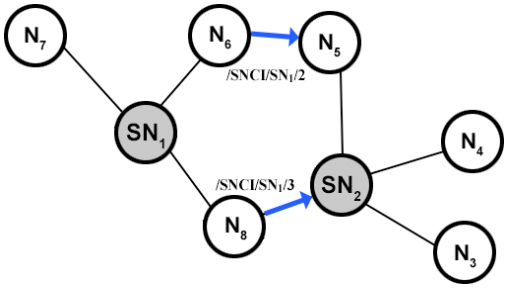
3. In Fig. 6.2d, when the SNCI message of SN_1 reaches SN_2, SN_2 replies with a Supernode Connection Data (SNCD) message. Then, SN_2 drops any other



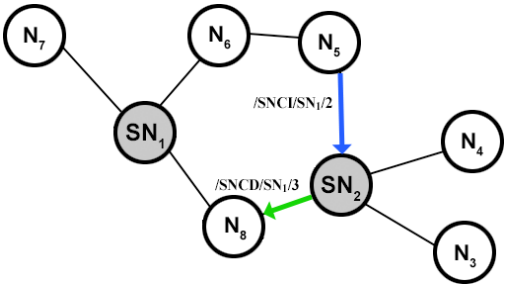
(a) A topology to explain the CDS construction algorithm



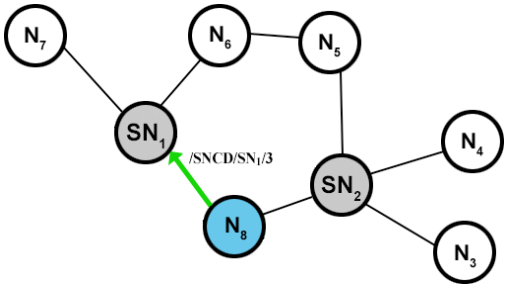
(b) SN_1 transmits an SNCI



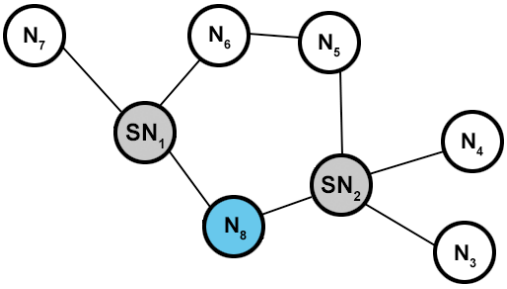
(c) SN_1's neighbors forward the received SNCIs



(d) When SN_1's SNCI reaches SN_2, it replies with an SNCD



(e) Node N_8 appoints itself as an SN and updates its *snFaceList*



(f) SN_1 updates its *snFaceList*

Figure 6.2: The CDS construction algorithm

received SNCI messages issued by SN_1, in this case, the one that is received from node N_5 because SN_2 has already replied to SN_1's SNCI message before.

4. Every node that gets an SNCD message further forwards it, until it reaches the issuing SN. Furthermore, in Fig. 6.2e, as the SNCD message has reached node N_8, N_8 appoints itself as an SN and stores the face through which the SNCD message was received in its *snFaceList*.

5. In Fig. 6.2f, when SN_1 receives the SNCD message, it sets its *cFlag* to *true* status and stores the face over which the SNCD message was received in its *snFaceList*. SN_1 and SN_2 are now connected through N_8.

We have to maintain DS and CDS in the following cases 1) when a new node joins the network, or 2) when a node or a link fails. If a new node joins the network, the maintenance of DS or CDS is quite simple. The new node simply requires to run the DS and CDS construction algorithms described in Sections 6.2.1 and 6.2.2 to join the DS or the CDS, respectively. To maintain DS and CDS when failures happen, we consider two cases 1) regular failures (a regular node or a link connecting two regular nodes fails), 2) SN failures (an SN or a link connected to an SN fails). When regular failures happen, if the entire graph is still connected, the DS or CDS will not change. Nevertheless, if an SN or a link connected to an SN fails, the node (or nodes) that were connected to the SN need to be aware of this failure so that they run again the DS and CDS construction algorithms (Sections 6.2.1 and 6.2.2).

So far, we have described distributed algorithms for DS and CDS construction. The proposed DS construction algorithm requires fewer message exchanges than our CDS construction algorithm. However, when the network topology is clustered using a DS, SNs might not be directly connected. Hence, it requires multi-hop communication through regular nodes so that SNs can reach each other. Therefore, the intuition is that DS-based inter-domain routing requires more bandwidth overhead than CDS-based inter-domain routing. Our algorithm only uses local communication with one-hop neighbors to build a DS.

For routing, we consider two cases in Sections 6.3 and 6.4, respectively: 1) L-SCN selects SNs using our DS construction algorithm described in Section 6.2.1, and 2) L-SCN selects SNs using our CDS construction algorithm described in Section 6.2.2.

Table 6.1: SRAD structure

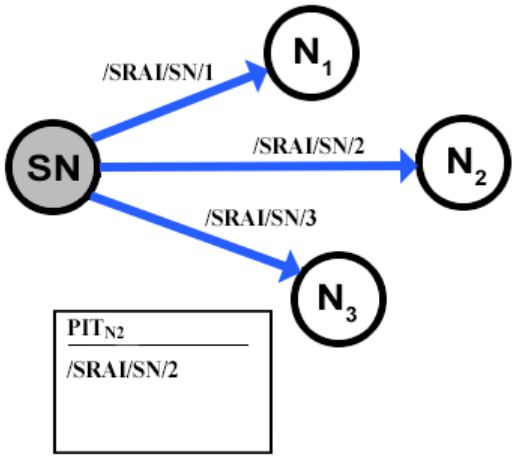
Timestamp	Generation time
Available service names	BF bit vector, salt count value.
Available resources	Available CPU, GPU, RAM,

6.3 Routing in a Dominating Set

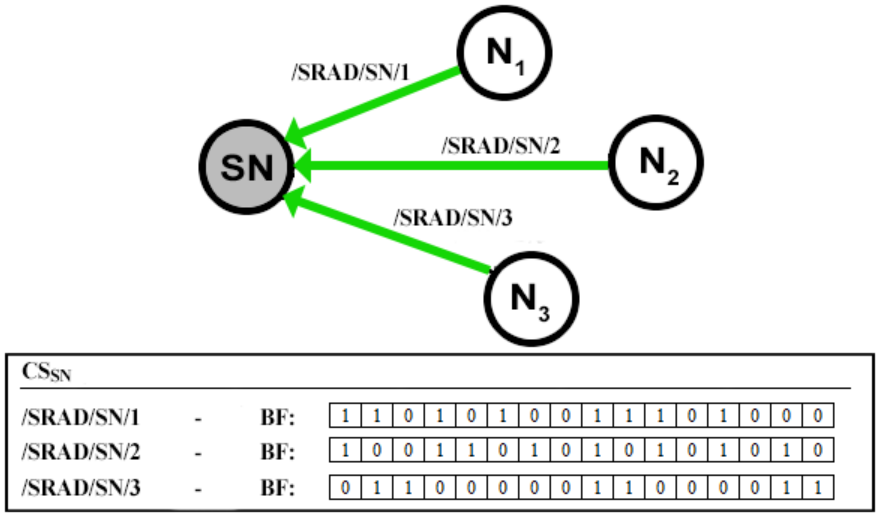
The proposed DS and CDS-based clustering algorithms, presented in Section 6.2, complement L-SCN by efficiently clustering the network topology, which is a requirement of L-SCN. After clustering the network topology, we focus on intra-domain and inter-domain routing. At each domain, the associated SN is responsible for both intra-domain and inter-domain routing. Intra-domain routing consists of routing service requests towards service provider(s), which are in the same domain, while inter-domain routing means routing a service request from a domain towards service provider(s) in another domain. Therefore, before routing, each SN needs to know the services and resources provided in its domain. According to this information, an SN decides whether a requested service could be provided within the domain or whether it requires inter-domain routing. Note that intra-domain routing is identical for both DS-based and CDS-based clustered network topologies. However, inter-domain routing is different for DS-based and CDS-based clustered networks.

6.3.1 Service and Resource Discovery

To acquire knowledge about the provided services and resources (e.g., CPU, RAM), each SN asks the nodes in its domain about their available services and resources using the pull mechanism that we illustrate in Fig. 6.3. Fig 6.3a shows that the SN sends over each face i a Service and Resource Availability Interest (SRAI) message with name $/SRAI/SN/i$ to pull information about available services and resources. Each regular node (N_1, N_2, or N_3) receives an SRAI message and stores in its PIT this message as well as the information about the reception face for it (see Fig. 6.3a). Then, each regular node replies with a Service and Resource Availability Data (SRAD) message, with the same name of its corresponding SRAI message that carries a BF containing its available service names as well as a field containing information about its available processing and memory resources. Table 6.1 shows the structure of an SRAD message, which includes the generation time of the SRAD message, a bit vector, and a salt count value that are needed to retrieve the BF containing available service



(a) SRAI packet broadcast to pull available services and resources



(b) SRAD packet forward by regular nodes containing BFs and resource information

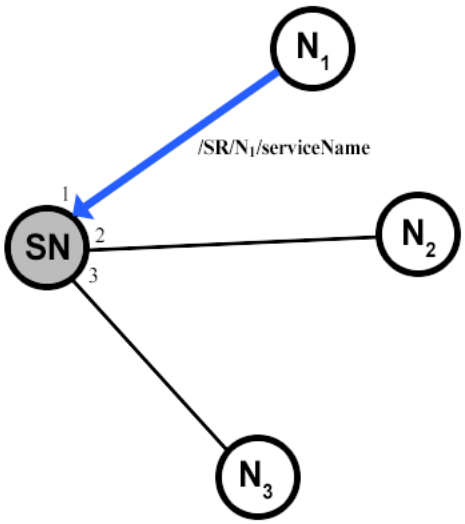
Figure 6.3: Pulling service and resource availability

names, and available processing and memory resources.

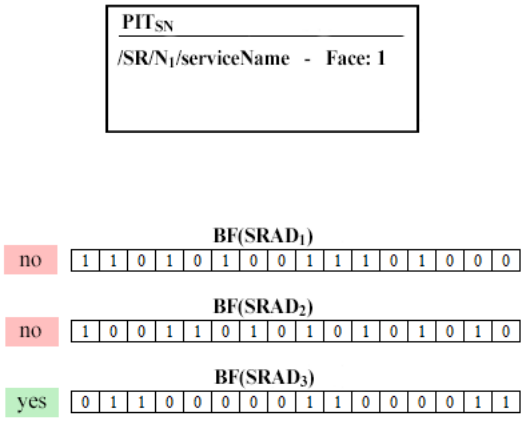
As Fig. 6.3b shows, the SN stores the received SRAD messages as well as the information about their reception faces in its content store.

6.3.2 Intra-Domain Routing

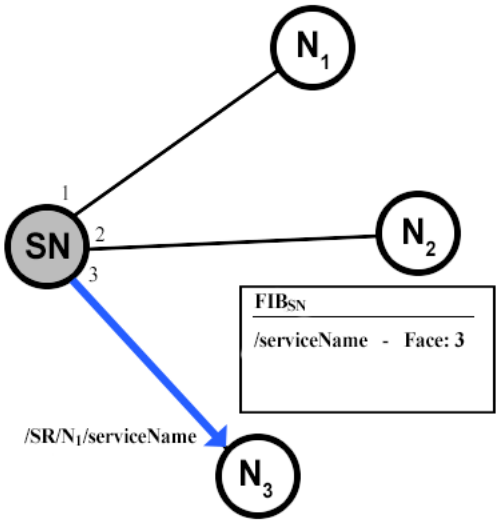
Fig. 6.4 describes FIB population and intra-domain routing. When node N_1 requires a service, it sends a Service Request (SR) message SR_1 with name $/SR/N_1/serviceName$ to the SN (see Fig. 6.4a). When SN receives message SR_1, it stores this message in the PIT and checks the name prefix */serviceName* (last name component of message SR_1) against all the BFs of the stored SRAD messages (see Fig. 6.4b). Since only the BF of $SRAD_3$ claims to contain name prefix */serviceName*, SN assigns in the FIB face ID 3 as the next hop face ID for */serviceName* and forwards message SR_1 over this face towards node N_3 (Fig. 6.4c). If multiple BFs claim to contain a service name, SN checks the associated SRAD messages for resource availability and forwards SR_1 to the node with the highest available resources. When node N_3 receives message SR_1, it replies with Service Data (SD) message(s) with name(s) $/SD/N_3/serviceName/sequenceNumber$ if it tends to provide the demanded service. Otherwise, node N_3 sends a Service Provision Refusal (SPR) message with name $/refusal/N_3/serviceName$ to the SN. If SN receives such an SPR message from N_3, it removes the FIB entry for name prefix */serviceName* and checks whether another regular node is available in the domain to provide the demanded service. If no node in the domain provides a certain service, the requested service requires inter-domain routing to be retrieved. In Fig. 6.4b, when SN checks the BFs of its stored SRAD messages to reply to SR_1, a false positive error might happen. For example, if a false positive error happens at $BF(SRAD_1)$, SN forwards message SR_1 to N_1. Since N_1 does not provide the service requested by message SR_1, N_1 returns an SPR message to deny the provision of the requested service. Nevertheless, false negative errors are impossible using BFs. Thus, SN will anyway forward message SR_1 towards the correct service provider, i.e., node N_3. Therefore, the proposed intra-domain routing mechanism is robust to false positive errors.



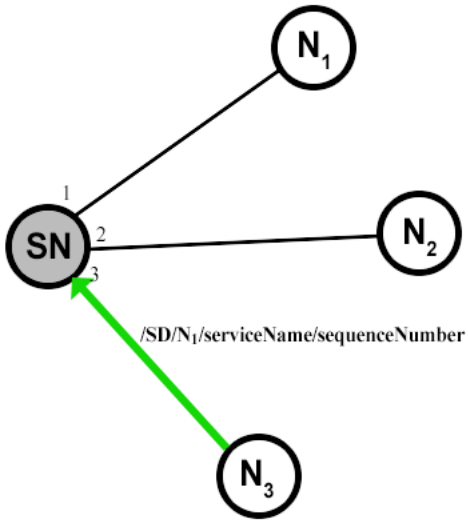
(a) Service request from N_1 to SN



(b) BF check at SN for name prefix $/serviceName$



(c) FIB population and forwarding SR to N_3



(d) Service data follows the reverse path of the service request

Figure 6.4: Intra-domain routing

6.3.3 Inter-Domain Routing for DS-based Clustering

If an SN receives an SR message, it first checks the name of the SR message against all the BFs of the SRAD messages stored in its CS table. If one of the BFs contains the name of the SR message, the SR message is routed inside the domain (intra-domain routing). Otherwise, the SR message is flooded until it reaches another SN (As explained in Section 6.2, the distance between two SNs in a DS does not exceed three hops). When another SN receives the SR message, it checks whether the service requested by message SR is provided in its domain. If the requested service is not available, the SN again floods the SR message until it reaches an SN that its domain provides the requested service. In Section 6.5, we will observe that, due to these SR message floodings, DS-based routing requires much more bandwidth overhead for routing SR messages compared to CDS-based routing.

6.4 Routing in a Connected Dominating Set

When CDS is used for clustering, SNs are directly connected. Therefore, inter-domain routing is different than DS-based routing. Fig. 6.5 shows an example CDS of four SNs SN_1, SN_2, SN_3 and SN_4 inside the dotted area. We assume that each SN records the face(s) over which it is connected to other SNs in a vector called *snFaceList*. For example, $snFaceList(SN_2) = 1,2,4$. In Fig. 6.5, when SN SN_2 receives service request message SR with name SRN from node N_4, which is asking for a service not provided in the domain of SN SN_2, this SN checks its FIB for name SRN. If no FIB entry is found, SN_2 forwards message SR over all the faces 1, 2 and 4 that exist in its *snFaceList*. Later, if SN_2 receives an SD message with name SRN over face 2, it populates the FIB for name SRN with next hop face 2. This FIB population helps SN_2 not to forward future service requests with name SRN over all the faces of its snFaceList.

6.5 Performance Evaluation

To evaluate the performance of our protocols, we have implemented them in ndnSIM2.1 [56]. We compare our protocols with NDN *multicast* forwarding strategy [8].

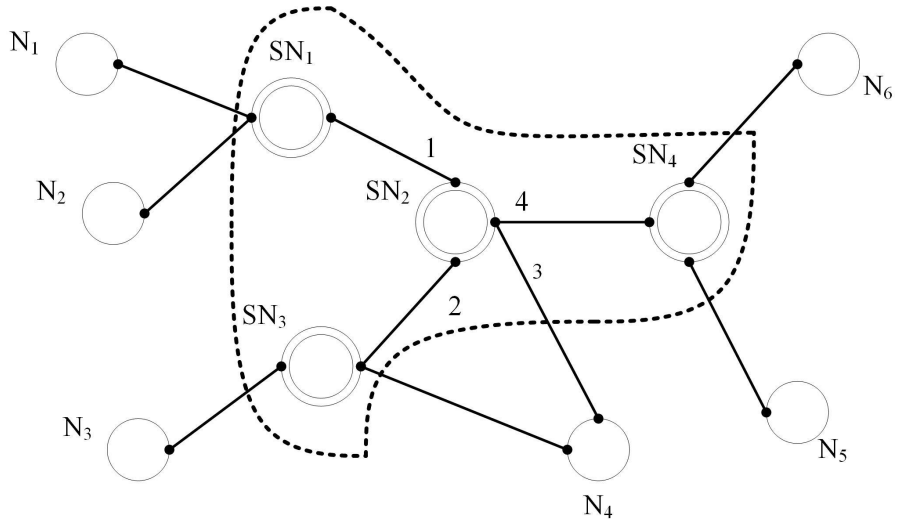


Figure 6.5: Inter-domain routing for CDS-based clustering.

6.5.1 Simulation Settings

For performance evaluation, we use GEANT [2], and Rocketfuel topologies [7]. GEANT topology consists of 40 routers. From Rocketfuel topology traces [7], we use three topologies with sizes 163, 624, and 1545 routers called RF-163, RF-624, and RF-1545, respectively. We place 10 service providers and 20 clients randomly in all the topologies. We assume that service providers process the received service requests and the processing time of a service request is uniformly distributed between 100 and 2000 milliseconds. If a client sends an SR message at time t_1 and receives the processed SD at time t_2, the client considers $t_2 - t_1$ the *service retrieval time*. Similar to [31], we assume that SNs pull the available services and resources every 10 seconds, while clients request services using a random function with exponential distribution with the mean value of 2 seconds. We assume that the service universe (i.e., the set of all available services) consists of 1000 unique services. The results are averaged over ten simulations and the reported mean values have 95% confidence intervals.

In the following, we analyze the performance of our protocols according to three metrics 1) the required bandwidth overhead for clustering, 2) the required bandwidth overhead for routing, and 3) average service retrieval time.

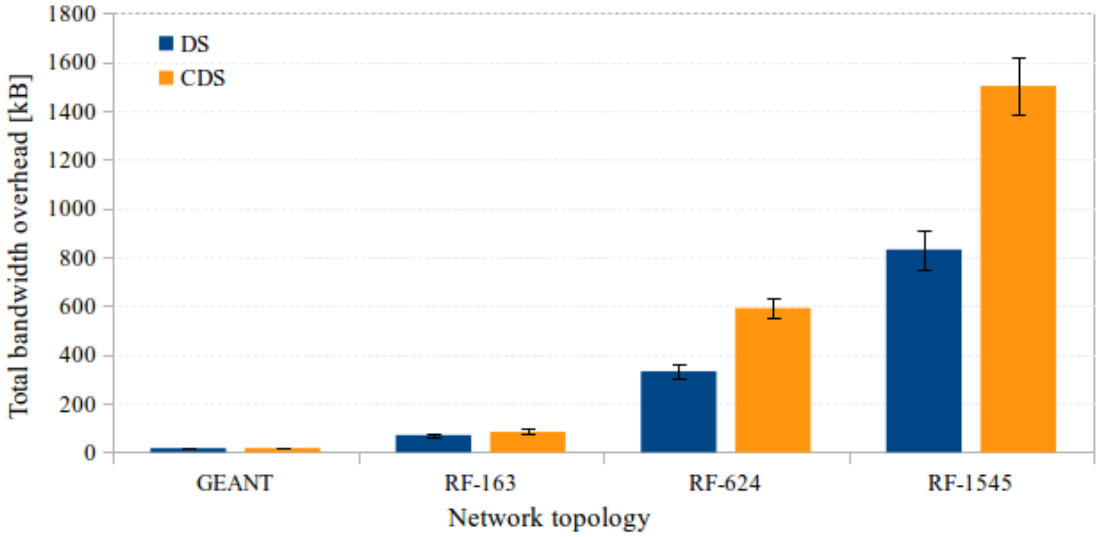
6.5.2 Bandwidth Overhead of Clustering

In Fig. 6.6a, we show the total bandwidth used for DS and CDS construction. From this evaluation, we can see that when topology size (i.e., the number of nodes) increases, the number of messages exchanged for DS and CDS construction also increases. Further, Fig. 6.6a shows that for all of the considered topologies, the proposed CDS construction algorithm requires more bandwidth overhead than the proposed DS construction algorithm because it builds on top of the DS construction algorithm. Nevertheless, from Fig. 6.6a we observe that for RF-624 and RF-1545, the proposed CDS construction algorithm requires more than twice the bandwidth overhead that the DS construction algorithm requires. This result confirms our intuition that when topology size increases, the required bandwidth overhead for CDS construction compared to that needed for DS construction also increases.

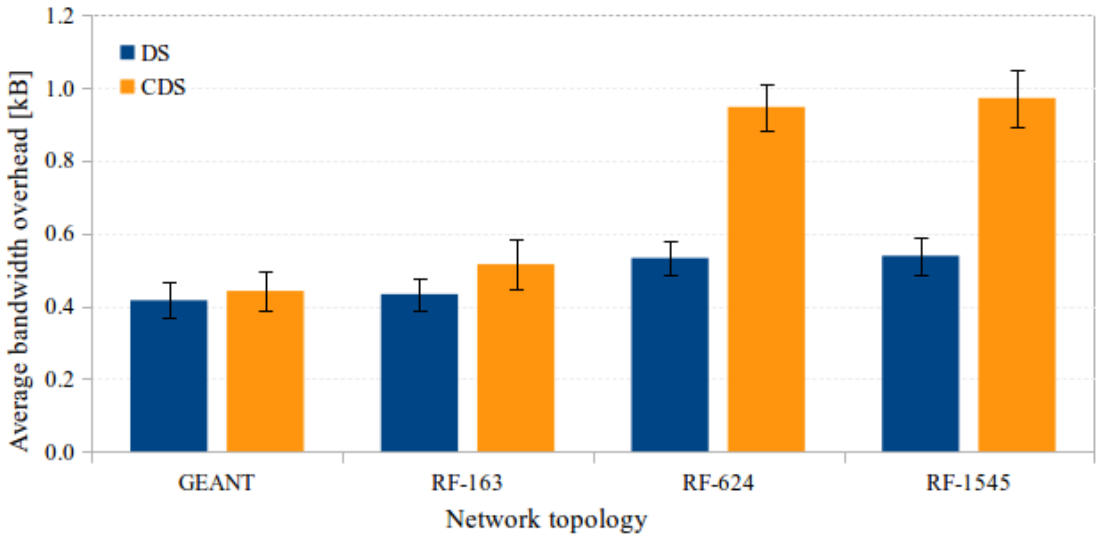
In Fig. 6.6b, we present the average bandwidth used for DS and CDS construction, i.e., the total bandwidth used for DS and CDS construction divided bt the number of network nodes. In Figs 6.6a and 6.6b, if we focus on each of the considered topologies separately and compare DS-based and CDS-based clustering algorithms, we can note the maximum difference between the orange and blue curves if RF-624 is used. This is an important observation, which confirms the impact of topology structure on the complexities of DS and CDS construction algorithms in terms of the required bandwidth overhead.

6.5.3 Bandwidth Overhead of Routing

In Fig. 6.7, we compare our DS-based and CDS-based routing protocols with NDN *multicast* forwarding strategy [8] in terms of bandwidth used for routing SR messages. For all the considered topologies, we expected less overhead for service request routing using a CDS. However, interestingly, we observe from Fig. 6.7 that for the GEANT topology, we see less overhead for service request routing when a DS is used. This result confirms the higher complexity of CDS-based routing compared to DS-based routing in terms of bandwidth overhead using small topologies. For larger topologies, the results are as expected and the CDS-based routing requires much less bandwidth overhead for routing service requests. The CDS-based routing protocol uses up to 39% less overhead to route service request messages in the larger topologies.



(a) Total bandwidth overhead for clustering algorithms



(b) Average transmitted bytes per node for clustering algorithms

Figure 6.6: Results in terms of bandwidth overhead for DS and CDS construction.

6.5.4 Average Service Retrieval Time

In Fig. 6.8, we show results in terms of average service retrieval time, i.e., the average time that a client experiences from the time it issues an SR message to the time it retrieves the requested service. From Fig. 6.8 we can see that NDN MultiCast (NDN-MC) forwarding strategy has always significantly higher average service retrieval

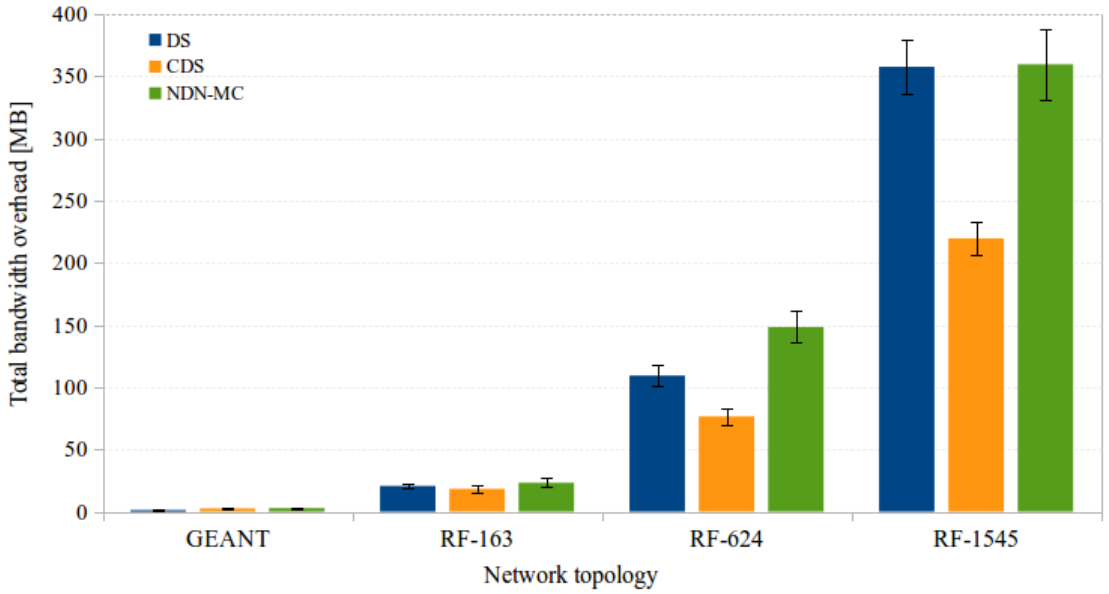
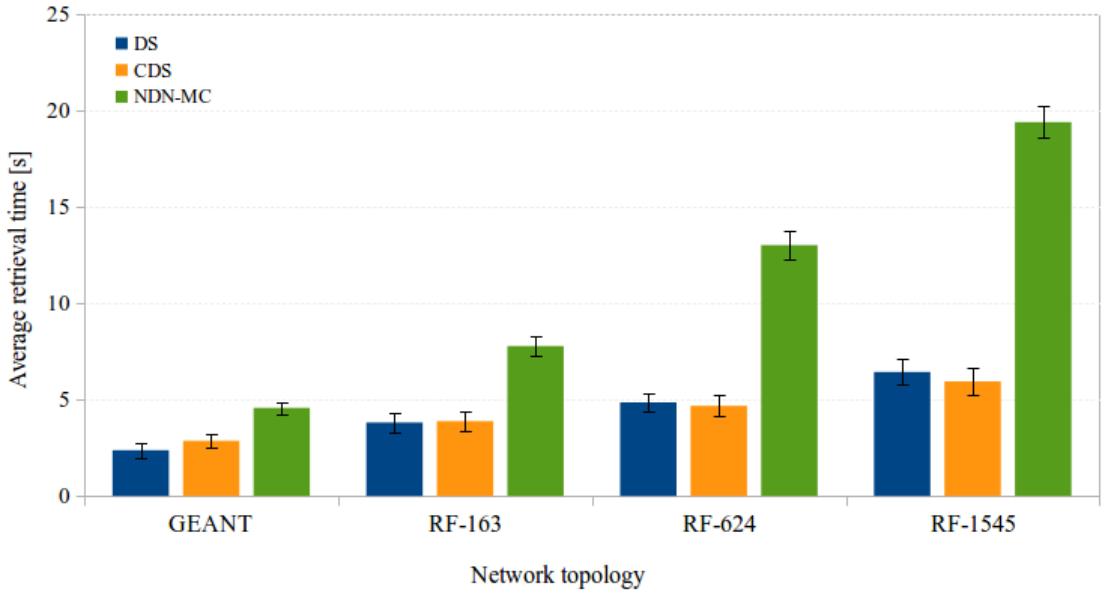


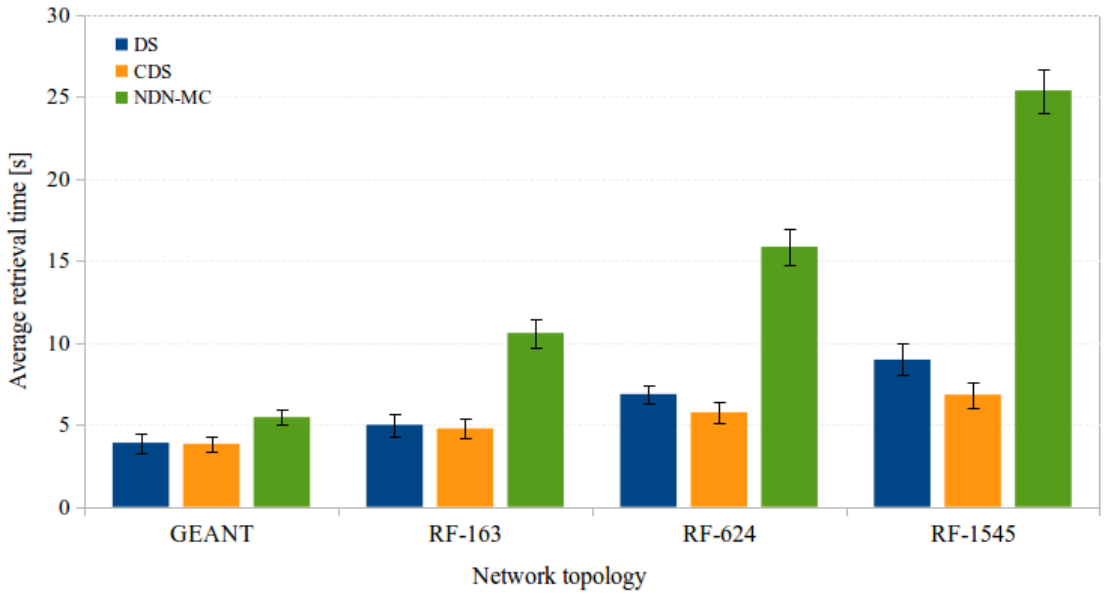
Figure 6.7: Total bandwidth overhead for service request routing

time than both DS and CDS-based routing. The reason is that NDN-MC floods SR messages, which leads to overloading many service providers. Consequently, when the NDN-MC strategy is employed, many SR messages are queued at overloaded service providers until the service providers become idle to process them. This results in a 69% shorter service retrieval time for our CDS approach compared to NDN-MC. For small topologies, like GEANT, the DS approach performs around 17% faster, while for large topologies, like RF-1545, the CDS approach performs 8% faster.

From Fig. 6.8b, we observe the impact of restricted link capacities (i.e., when all the links have only 10% of their original capacities) on average service retrieval time. Fig. 6.8b makes clear that the bandwidth-hungry nature of the NDN-MC forwarding strategy has a considerable impact on average service retrieval time. Overall, we observe similar performance for DS and CDS-based routing in terms of average service retrieval time, while CDS-based routing performs slightly better than DS-based routing when there are tight bandwidth resources, which is because of higher bandwidth requirements for DS-based routing. Also, we note an even higher gain, compared to NDN-MC with reduced link capacity. The service retrieval takes up to 73% less time with our CDS-based routing. The CDS approach is also up to 24% faster than the DS approach with reduced link capacities.



(a) Average service retrieval time with full link capacities



(b) Average service retrieval time with reduced link capacities

Figure 6.8: Results in terms of service retrieval time.

6.6 Conclusions

In this Chapter, we studied intra-domain and inter-domain routing of service requests in SCNs. We proposed fully distributed algorithms for constructing DS and CDS to

select SNs in the network topology. Next, we leveraged SNs to implement intra and inter-domain service and resource discovery for SCNs. For service discovery, we used BFs to reduce the required bandwidth overhead for service availability advertisements. Using the information collected during the proposed service and resource discovery method, we implemented BF-based and resource-aware intra-domain and inter-domain routing protocols to route service requests.

We assessed the performance of our routing protocols for various topologies of different sizes. From the results, we observed that with larger topologies, DS construction requires much less bandwidth than CDS construction. We compared the proposed DS-based and CDS-based routing protocols with NDN-MC. The results showed that the DS-based routing protocol requires less bandwidth overhead for routing service requests than the NDN-MC forwarding strategy, while the CDS-based routing protocol requires much less bandwidth overhead for service request routing compared to DS-based routing protocol. Finally, we observed that both DS-based and CDS-based routing protocols outperform NDN-MC in terms of service retrieval time. To the best of our knowledge, this Chapter is the first work that leverages DS and CDS concepts for BF-based and resource-aware intra and inter-domain routing in SCNs.

7

Conclusions

7.1 Summary

In the Introduction, we posed three RQs related to NDN. We mentioned that RQ 1 investigates how to route Interest messages in NDN. To answer RQ 1, In Chapter 3, we presented a routing protocol called push-based BFR. RQ 2 investigates how to reduce the required bandwidth and storage resources for push-based BFR's content advertisement messages. To address RQ 2, in Chapter 4, we described pull-based BFR as another BF-based routing protocol for NDN and showed that pull-based BFR requires significantly less bandwidth and storage resources for content advertisements compared to push-based BFR. We posed RQ 3 related to reducing content retrieval delay in NDN. To answer RQ 3, in Chapter 5, we presented a protocol based on multi-session NC for content retrieval and showed that it achieves lower content retrieval delay compared to push-based and pull-based BFR.

In the last part of this thesis, we studied routing in SCN [21]. In the Introduction, we mentioned that L-SCN [31] was proposed as a routing architecture for SCN, which addresses service discovery, resource discovery, and load balancing. However, L-SCN

requires algorithms to select SNs before routing. Therefore, RQ 4 investigates how to select SNs in L-SCN. In Chapter 6, we suggested to create DS and CDS over the network topology and to select the dominator nodes as SNs. Therefore, we presented distributed algorithms that create DS and CDS over the network topology and select the SNs. Then, we implemented BF-based intra-domain and inter-domain routing protocols for SCN.

In the next Sections, we first describe the main contributions of this thesis. Then, we propose some future research directions.

7.2 Main Contributions

Routing protocols can route Interest messages by looking up the FIB tables. Therefore, FIB population is a requirement for routing protocols. For FIB population, push-based BFR permits servers to compactly represent the sets of their provided content object names using BFs, and propagate the calculated BFs for content advertisements. When clients and routers receive the content advertisement BFs, they can check the names of the received Interest messages against the content advertisement BFs, populate the FIBs, and, then, forward the Interest messages to retrieve the requested content objects. We compared push-based BFR with flooding, shortest path, and COBRA [72]. We observed from the results that push-based BFR significantly outperforms shortest path routing in terms of content advertisement overhead. Further, push-based BFR outperforms flooding and COBRA in terms of communication overhead and the average round-trip delay. Push-based BFR requires significantly less storage resources for storing BFs compared to the storage space that COBRA needs for storing SBFs. Therefore, if network nodes have memory space limitations (e.g., sensors in IoT scenarios with constrained nodes), it is more appropriate to use push-based BFR than COBRA.

Push-based BFR is an efficient routing protocol with the following features: 1) fully distributed and content-oriented design, 2) topology oblivious, 3) robust to topology changes. Nevertheless, due to the push-based content advertisements strategy of push-based BFR, the bandwidth and storage resources required for content advertisements linearly grows with the content universe size. Clients only demand a small number of content objects from the entire content universe. Thus, it is sufficient if servers only advertise the names of the demanded content objects. We designed pull-

based BFR as a routing protocol that only advertises the demanded content objects. To make servers aware of the demanded content objects, clients calculate BFs containing the names of their Interest messages, put the BFs in CAR messages, and send the CAR messages towards upstream. Routers that receive CAR messages, aggregate their BFs using a practical BF aggregation strategy and continue sending them until they reach the servers. Finally, when servers receive the CAR messages, they calculate BFs of the demanded content object names, put the BFs in CA messages, and propagate the CA messages for content advertisements. When clients and routers receive the CA messages, they can populate the FIBs and route the Interest messages. We compared pull-based BFR with push-based BFR and FaR [68] protocols. The results showed that pull-based BFR significantly outperforms push-based BFR in terms of the required bandwidth and storage resources for content advertisements. Further, from the results, we observed that pull-based BFR is more robust to BF false positive reports than push-based BFR, and pull-based BFR outperforms both push-based BFR and FaR [68] in terms of average round-trip delay.

In Chapter 5, we presented a multi-session NC protocol that benefits from the received BFs sent during the pull-based content discovery phase to select network codes of the requested Data messages. Further, to manage the multi-session codeblock size, we let the nodes collaborate by sending local feedbacks about their available capacity for accepting new variables in their equation systems, a BF containing their decoded variables that they do not need to receive in linear combinations, and a BF containing the variables that are involved in the equation system. These feedback messages allow nodes to assign utility values to the available variables and solve an instance of a linear program to select network codes.

In Chapter 6, we developed a routing protocol for SCN consisting of two phases. In the first phase, the proposed protocol presented distributed and localized algorithms to construct DS and CDS over the network topology. Running the algorithms the dominator nodes play the role of SNs and the dominated nodes that are connected to an SN are considered as the domain nodes. Then, we proposed BF-based intra-domain and inter-domain routing protocols for both DS-based and CDS-based clustered networks. We compared our DS-based and CDS-based routing protocols with NDN multicast strategy and observed from the results that 1) the bandwidth overhead for creating DS and CDS increases with topology size, 2) for large topologies, the proposed CDS-based routing protocol requires drastically less bandwidth overhead than both DS-based routing protocol and NDN-MC, 3) CDS-based routing protocol

achieves slightly lower service retrieval time than DS-based routing protocol, while both CDS-based routing protocol and DS-based routing protocol require much less service retrieval time than NDN-MC strategy.

7.3 Future Research Directions

Although in this thesis we designed and evaluated BF-based routing protocols for NDN and SCN, and a cooperative multi-session NC protocol for content retrieval in NDN, as always, there are possibilities for extending the scope of the work. In this Section, we mention some of the future research directions.

A future research direction could be related to the security of BF-based routing. Using push-based BFR, origin servers propagate BFs for content advertisements. However, malicious network nodes might pretend to be content providers and perform an attack by injecting bogus content advertisement messages. To cope with this attack, it is required to integrate security mechanisms for authenticating origin servers. Origin servers might sign their content advertisement messages to indicate their authenticity. When an origin server signs a content advertisement message, it has to share its public key with the network nodes so that the network nodes can verify its signature. Origin servers can share their public keys by storing them into a centralized server. However, the centralized server will be a single point of failure. An alternative approach could distribute the public keys over multiple servers that form a distributed system. Nevertheless, each server might attack individually by modifying the stored public keys. To cope with this problem, blockchain [57] is proposed as a distributed system that any change in the stored content of a blockchain node has to be reported to the other blockchain nodes and requires their approval. Therefore, blockchain provides transparency to all the activities of the blockchain nodes. Due to these appealing features of the blockchain, future research works could attempt to design novel protocols that leverage a blockchain [57] to store origin servers' public keys.

In pull-based BFR, when a router receives different CAR/CA messages, the router could make a union of the received CAR/CA messages and aggregate them. However, a malicious router might perform other bitwise operations, e.g., exclusive or (XOR), on the received BFs to pollute them. When a malicious router pollutes a BF of a CAR/CA message and forwards it, the polluted BF might be aggregated with the BFs of other CAR/CA messages and makes them also polluted. Therefore, it is important to design

security mechanisms to detect pollution attacks on BFs and to discard the polluted BFs. To the best of our knowledge, previous research works have investigated pollution attacks with arithmetic operations (e.g, NC) [9, 62], but there is a gap in the literature related to pollution attacks with bitwise operations, which could be a research topic for the future.

We proposed a multi-session NC protocol that permits the routers to linearly combine the network coded messages. However, a router could create a bogus network coded message, which is not a linear combination of the original messages, and combine the bogus network coded message with the received legitimate network coded messages to perform a pollution attack. It is critical to cope with pollution attacks on network codes because when a node forwards a polluted network coded message, the node that receives the polluted network coded message combines it with other network coded messages, and, therefore, the other network coded messages will be also polluted. To cope with pollution attacks when NC is used, the work in [9] proposes to add Message Authentication Codes (MAC) to the original messages that have the closure property under linear NC operations. [9] assumes that the MACs of the original messages are shared with the network nodes so that they can validate the MACs of the network coded messages. However, sharing the MACs of all original messages would require significant bandwidth and storage resources. Future research works might focus on designing security schemes to tackle pollution attacks on network codes that require less bandwidth and storage resources.

To select SNs, we suggested fully distributed methods that leverage DS and CDS concepts. However, there are open security problems to investigate in the future. For example, malicious clients could flood service requests to attack the SNs. Controlling the service request injection rate could mitigate this problem. However, future research works might provide more effective solutions to this problem. Another research opportunity is to investigate how to cope with malicious SNs that might drop the received service requests or might route them towards wrong service providers. L-SCN requires to integrate trust mechanisms to significantly reduce the risk of malicious activities by SNs.

8

List of Acronyms

BF Bloom Filter

BFR Bloom Filter-based Routing

CA Content Advertisement

CAI Content Advertisement Interest

CAR Content Advertisement Request

CD Clustering Data

CII Clustering Initializing Interest

CDS Connected Dominating Set

CCN Content-Centric Networking

COBRA COntent-oriented intra-domain Bloom filter-based Routing Algorithm

CS Content Store

CU Content Universe

DONA Data-Oriented Network Architecture

DHT Distributed Hash Table

DS Dominating Set

DASH Dynamic Adaptive Streaming over HTTP

FDM Feedback Data Message

FIM Feedback Interest Message

FaR Flooding-assisted Routing

FIB Forwarding Information Base

ICN Information-Centric Networking

IoT Internet of Things

IP Internet Protocol

LFU Least Frequently Used

LNC Linear Network Coding

LRU Least Recently Used

LSA Link-State Advertisement

L-SCN Layered-Service Centric Networking

LPM Longest Prefix Matching

MCDS Minimum Connected Dominating Set

MDS Minimum Dominating Set

NDN Named Data Networking

NLSR Named-data Link State Routing

NFaaS Named Function as a Service

NFD NDN Forwarding Daemon

NFN Named Function Networking

NC Network Coding

NetInf Network of Information

NREN National Research and Education Network

PIT Pending Interest Table

PRLNC Prioritized Random Linear Network Coding

PURSUIT Publish Subscribe Internet Technology

RQ Research Question

RLNC Random Linear Network Coding

SVC Scalable Video Coding

SCN Service-Centric Networking

SD Service Data

SPR Service Provision Refusal

SR Service Request

SN Super Node

SNCD Supernode Connection Data

SNCI Supernode Connection Interest

SSD Supernode Selection Data

SSI Supernode Selection Interest

SRAD Service and Resource Availability Data

SRAI Service and Resource Availability Interest

SBF Stable Bloom Filter

URL Uniform Resource Locator

Bibliography

[1] "Content-Centric Networking Project." http://www.ccnx.org/.

[2] "The GEANT Network," http://www.topology-zoo.org/dataset.html, accessed: 2016-07-25.

[3] "Mobility First Project." http://mobilityfirst.winlab.rutgers.edu/.

[4] "Named Data Networking Project." http://www.named-data.net/.

[5] "Ns-3: a discrete-event network simulator for internet systems," https://www.nsnam.org/, accessed: 2019-07-20.

[6] "ns-3 Direct Code Execution (DCE) Manual," http://www.topology-zoo.org/dataset.html.

[7] "Rocketfuel: An ISP Topology Mapping Engine," https://research.cs.washington.edu/networking/rocketfuel/, accessed: 2019-07-20.

[8] A. Afanasyev *et al.*, "NFD Developer's Guide," Named Data Networking project, Technical Report NDN-0021 Revision 7, Oct. 2016.

[9] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-based Integrity for Network Coding," in *Applied Cryptography and Network Security*, M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 292–305.

[10] B. Ahlgren, M. D'Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone, "Design Considerations for a Network of Information," in *Proc. of ACM international Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, Madrid, Spain, 2008, pp. 1–6.

[11] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[12] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Trans. Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[13] K. M. Alzoubi, P.-J. Wan, and O. Frieder, "Message-Optimal Connected Dominating Sets in Mobile Ad-hoc Networks," in *Proc. of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, Lausanne, Switzerland, 2002, pp. 157–164.

[14] M. Badov, A. Seetharam, J. Kurose, V. Firoiu, and S. Nanda, "Congestion-aware Caching and Search in Information-Centric Networks," in *Proc. of the ACM 1st international conference on Information-Centric Networking*, Paris, France, Sep. 2014, pp. 37–46.

[15] L. Berger, I. Bryskin, A. Zinin, and R. Coltun, "The OSPF Opaque LSA Option," *RFC*, vol. 5250, pp. 1–17, Jul. 2008.

[16] N. Blefari-Melazzi, A. Detti, M. Arumaithurai, and K. K. Ramakrishnan, "Internames: A Name-to-Name Principle for the Future Internet," in *Proc. of the 10th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, 2014, pp. 146–151.

[17] B. H. Bloom, "Space/time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[18] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-Preserving Symmetric Encryption," vol. 5479, pp. 224–241, 2009.

[19] E. Bourtsoulatze, N. Thomos, and P. Frossard, "Distributed Rate Allocation in Inter-Session Network Coding," *IEEE Trans. Multimedia*, vol. 16, no. 6, pp. 1752–1765, 2014.

[20] E. Bourtsoulatze, N. Thomos, J. Saltarin, and T. Braun, "Content-Aware Delivery of Scalable Video in Network Coding Enabled Named Data Networks," *IEEE Trans. Multimedia*, vol. 20, no. 6, pp. 1561–1575, 2018.

[21] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello, "Service-Centric Networking," in *Proc. of the IEEE International Conference on Communications (ICC) Workshops*, Kyoto, Japan, June 2011, pp. 1–6.

[22] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[23] M. Cardei, M. X. Cheng, X. Cheng, and D.-Z. Du, "Connected Domination in Multihop Ad hoc Wireless Networks," in *Proc. of the 6th Joint Conference on Information Science (JCIS)*, Mar. 2002, pp. 251–255.

[24] R. Chiocchetti, D. Perino, G. Carofiglio, D. Rossi, and G. Rossini, "Inform: A Dynamic Interest Forwarding Mechanism for Information-Centric Networking," in *Proc. of the ACM 3rd SIGCOMM workshop on Information-Centric Networking*, Hong Kong, China, Aug. 2013, pp. 9–14.

[25] P. A. Chou, Y. Wu, and K. Jain, "Practical Network Coding," in *Proc. of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, no. 1, 2003, pp. 40–49.

[26] M. E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transactions on Networking (TON)*, vol. 5, no. 6, pp. 835–846, Dec. 1997.

[27] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone, "MDHT: A Hierarchical Name Resolution Service for Information-Centric Networks," in *Proc. of the ACM SIGCOMM workshop on Information-Centric Networking (ICN)*, Toronto, Ontario, Canada, Aug. 2011, pp. 7–12.

[28] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of Information (NetInf)–An Information-Centric Networking Architecture," *Computer Communications*, vol. 36, no. 7, pp. 721–735, Apr. 2013.

[29] F. Deng and D. Rafiei, "Approximately Detecting Duplicates for Streaming Data Using Stable Bloom Filters," in *Proc. of the ACM SIGMOD international conference on Management of data*, Chicago, IL, USA, Jun. 2006, pp. 25–36.

[30] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network Coding: An Instant Primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63–68, Jan. 2006.

[31] M. Gasparyan, T. Braun, and E. Schiller, "L-SCN: Layered SCN Architecture with Supernodes and Bloom Filters," in *Proc. of the 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan 2017, pp. 899–904.

[32] M. Gasparyan, G. Corsini, T. Braun, E. Schiller, and J. Saltarin, "Session Support for SCN," in *2017 IFIP Networking Conference and Workshops*, Jun. 2017.

[33] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Trans. on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[34] T. Ho, M. Medard, J. Shi, M. Effros, and D. R. Karger, "On Randomized Network Coding," in *Proc. of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, no. 1, 2003, pp. 11–20.

[35] A. K. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "NLSR: Named-data Link State Routing Protocol," in *Proc. of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking*, Hong Kong, China, Aug. 2013, pp. 15–20.

[36] ICNRG, "Design Choices and Differences for NDN and CCNx 1.0 Implementations of Information-Centric Networking," https://icnrg.github.io/draft-icnrg-harmonization/draft-icnrg-harmonization-00.txt, ICNRG, Jul. 2017.

[37] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proc. of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, Rome, Italy, 2009, pp. 1–12.

[38] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. M. G. M. Tolhuizen, "Polynomial Time Algorithms for Multicast Network Code Construction," *IEEE Trans. Information Theory*, vol. 51, no. 6, pp. 1973–1982, 2005.

[39] A. W. Kazi, "Prefetching Bloom Filters to Control Flooding in Content-Centric Networks," in *Proc. of the ACM CoNEXT Student Workshop*, Philadelphia, Pennsylvania, USA, Nov. 2010, p. 22.

[40] R. Koetter and M. Médard, "An Algebraic Approach to Network Coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, 2003.

[41] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A Data-oriented (and Beyond) Network Architecture," in *Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Kyoto, Japan, Aug. 2007, pp. 181–192.

[42] M. Król and I. Psaras, "NFaaS: Named Function as a Service," in *Proc. of the 4th ACM Conference on Information-Centric Networking (ICN)*, Berlin, Germany, Sep. 2017, pp. 134–144.

[43] E. Kurdoglu, N. Thomos, and P. Frossard, "Scalable Video Dissemination with Prioritized Network Coding," in *Proc. of the IEEE International Conference on Multimedia and Expo (ICME)*, Barcelona, Spain, Jul. 2011, pp. 1–6.

[44] R. Laskar and H. Walikar, "On Domination-related Concepts in Graph Theory," in *Combinatorics and graph theory*, 1981, pp. 308–320.

[45] M. Lee, K. Cho, K. Park, T. T. Kwon, and Y. Choi, "SCAN: Scalable Content Routing for Content-aware Networking," in *Proc. of the IEEE international conference on Communications (ICC)*, Kyoto, Japan, Jun. 2011, pp. 1–5.

[46] C. Lenzen, Y.-A. Pignolet, and R. Wattenhofer, "Distributed Minimum Dominating Set Approximations in Restricted Families of Graphs," *Distributed Computing*, vol. 33, 2013.

[47] S. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," *IEEE Trans. Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[48] Z. Li, K. Liu, Y. Zhao, and Y. Ma, "MaPIT: An Enhanced Pending Interest Table for NDN With Mapping Bloom Filter," *IEEE Communications Letters*, vol. 18, no. 11, pp. 1915–1918, Nov. 2014.

[49] H. Liu, X. De Foy, and D. Zhang, "A Multi-level DHT Routing Framework with Aggregation," in *Proc. of the ACM 2nd edition of the ICN Workshop on Information-Centric Networking*, Helsinki, Finland, Aug. 2012, pp. 43–48.

[50] A. Marandi, T. Braun, K. Salamatian, and N. Thomos, "BFR: A Bloom Filter-based Routing Approach for Information-Centric Networks," in *Proc. of the 16th International IFIP Networking Conference*, Stockholm, Sweden, Jun. 2017, pp. 1–9.

[51] ——, "Pull-based Bloom Filter-based Routing for Information-Centric Networks," in *Proc. of The 16th IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan. 2019, pp. 1–7.

[52] ——, "Network Coding-based Content Retrieval based on Bloom Filter-based Content Discovery for ICN," in *Proc. of the 54th IEEE Conference on Communications (ICC)*, Dublin, Ireland, Jun. 2020.

[53] A. Marandi, V. Hofer, M. Gasparyan, T. Braun, and N. Thomos, "Bloom Filter-based Routing for Dominating Set-based Service-Centric Networks," in *Proc. of the IEEE/IFIP Network Operations and Management Symp. (NOMS)*, Budapest, Hungary, Apr. 2020.

[54] A. Marandi, M. F. Imani, and K. Salamatian, "Optimization of Bloom Filter Parameters for Practical Bloom Filter-based Epidemic Forwarding in DTNs," *CoRR*, vol. abs/1208.3871, 2012. [Online]. Available: http://arxiv.org/abs/1208.3871

[55] ——, "Practical Bloom Filter-based Epidemic Forwarding and Congestion Control in DTNs: A Comparative Analysis," *Computer Communications*, vol. 48, pp. 98–110, 2014.

[56] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2.0: A New Version of the NDN Simulator for NS-3," *Technical Report NDN-0003*, Jan. 2015.

[57] A. A. Monrat, O. Schelén, and K. Andersson, "A survey of blockchain from the perspectives of applications, challenges, and opportunities," *IEEE Access*, vol. 7, pp. 117 134–117 151, 2019.

[58] M. A. Montemurro, "Beyond the Zipf–Mandelbrot Law in Quantitative Linguistics," *Physica A: Statistical Mechanics and its Applications*, vol. 300, no. 3-4, pp. 567–578, 2001.

[59] J. H. Mun and H. Lim, "Cache Sharing Using Bloom Filters in Named Data Networking," *Journal of Network and Computer Applications*, vol. 90, pp. 74–82, 2017.

[60] C. Muñoz, L. Wang, E. Solana, and J. Crowcroft, "I(FIB)f: Iterated Bloom Filters for Routing in Named Data Networks," in *Proc. of the International Conference on Networked Systems (NetSys)*, Göttigen, Germany, 2017, pp. 1–8.

[61] L. Muscariello, "CCNx Project," https://wiki.fd.io/view/Cicn#News, Jan. 2020.

[62] F. Oggier and H. Fathi, "An Authentication Code Against Pollution Attacks in Network Coding," *IEEE/Acm Transactions On Networking*, vol. 19, no. 6, pp. 1587–1596, 2011.

[63] S. Roos, L. Wang, T. Strufe, and J. Kangasharju, "Enhancing Compact Routing in CCN with Prefix Embedding and Topology-aware Hashing," in *Proc. of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture (MobiArch)*, Maui, Hawaii, USA, Sep. 2014, pp. 49–54.

[64] E. J. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, Best-effort Content Location in Cache Networks," in *Proc. of the IEEE International Conference on Computer Communications (INFOCOM)*, Rio de Janeiro, Brazil, Apr. 2009, pp. 2631–2635.

[65] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun, "NetCodCCN: A Network Coding Approach for Content-Centric Networks," in *Proc. of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, San Francisco, CA, USA, Apr. 2016, pp. 1–9.

[66] ——, "Adaptive video streaming with network coding enabled named data networking," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2182–2196, Oct. 2017.

[67] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H. 264/AVC Standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, 2007.

[68] J. Shi, E. Newberry, and B. Zhang, "On broadcast-based self-learning in named data networking," in *Proc. of the 16th International IFIP Networking Conference*, Stockholm, Sweden, Jun. 2017, pp. 1–9.

[69] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP– Standards and Design Principles," in *Proc. of the 2nd Annual ACM Conference on Multimedia Systems*, San Jose, CA, USA, Feb. 2011, pp. 133–144.

[70] D. G. Thaler and C. V. Ravishankar, "Using Name-based Mappings to Increase Hit Rates," *IEEE/ACM Trans. Netw.*, vol. 6, no. 1, pp. 1–14, Feb. 1998.

[71] N. Thomos, J. Chakareski, and P. Frossard, "Prioritized Distributed Video Delivery with Randomized Network Coding," *IEEE Trans. on Multimedia*, vol. 13, no. 4, pp. 776–787, 2011.

[72] M. Tortelli, L. A. Grieco, G. Boggia, and K. Pentikousisy, "COBRA: Lean Intra-domain Routing in NDN," in *Proc. of the IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan. 2014, pp. 839–844.

[73] D. Trossen and G. Parisis, "Designing and Realizing An Information-Centric Internet," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 60–67, Jul. 2012.

[74] D. Trossen, G. Parisis, K. Visala, B. Gajic, J. Riihijarvi, P. Flegkas, P. Sarolahti, P. Jokela, X. Vasilakos, C. Tsilopoulos *et al.*, "Pursuit Conceptual Architecture: Pinciples, Patterns and Sub-components Descriptions," *Technical Report NDN-0003*, May. 2011.

[75] C. Tschudin and M. Sifalakis, "Named Functions and Cached Computations," in *Proc. of the IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan 2014, pp. 851–857.

[76] C. F. Tschudin and M. Sifalakis, "Named Functions for Media Delivery Orchestration," in *Proc. of the 20th International Packet Video Workshop, PV*, San Jose, CA, USA, Dec. 2013, pp. 1–8.

[77] L. Wang, A. K. M. Mahmudul Hoque, C. Yi, A. Alyyan, and B. Zhang, "OSPFN: An OSPF Based Routing Protocol for Named Data Networking," *Technical Report NDN-0003*, pp. 1–15, Jul. 2012.

[78] L. Wang, S. Bayhan, J. Ott, J. Kangasharju, A. Sathiaseelan, and J. Crowcroft, "Prodiluvian: Understanding Scoped-Flooding for Content Discovery in Information-Centric Networking," in *Proc. of the ACM 2nd international conference on Information-Centric Networking*, San Francisco, CA, USA, Sep. 2015, pp. 9–18.

[79] L. Wang, O. Waltari, and J. Kangasharju, "MobiCCN: Mobility Support with Greedy Routing in Content-Centric Networks," in *Proc. of the IEEE Global Communications Conference (GLOBECOM)*, Atlanta, GA, USA, Dec. 2013, pp. 2069–2075.

[80] Y. Wang, K. Lee, B. Venkataraman, R. L. Shamanna, I. Rhee, and S. Yang, "Advertising Cached Contents in the Control Plane: Necessity and Feasibility," in *Proc. of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Orlando, FL, USA, Mar. 2012, pp. 286–291.

[81] J. Widmer, C. Fragouli, and J.-Y. Le Boudec, "Low-Complexity Energy-Efficient Broadcasting in Wireless Ad-Hoc Networks Using Network Coding," in *Proc. of the 1st Workshop on Network Coding, Theory, and Applications (NetCod)*, no. LCA-CONF-2005-016, Riva del Garda, Italy, 2005.

[82] Q. Wu, Z. Li, J. Zhou, H. Jiang, Z. Hu, Y. Liu, and G. Xie, "SOFIA: Toward Service-oriented Information-Centric Networking," *IEEE Network*, vol. 28, no. 3, pp. 12–18, May 2014.

[83] Y. Wu, P. A. Chou, and S.-Y. Kung, "Minimum-Energy Multicast in Mobile Ad Hoc Networks Using Network Coding," *IEEE Trans. on communications*, vol. 53, no. 11, pp. 1906–1918, 2005.

[84] Y. Xu, Y. Li, T. Lin, G. Zhang, Z. Wang, and S. Ci, "A Dominating-Set-based Collaborative Caching with Request Routing in Content-Centric Networking," in *Proc. of the IEEE International Conference on Communications (ICC)*, Budapest, Hungary, Jun. 2013.

[85] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A Survey of Information-Centric Networking Research," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 1024–1049, Feb. 2014.

[86] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "On The Role of Routing in Named Data Networking," in *Proc. of the 1st international conference on Information-Centric Networking*, Paris, France, Sep. 2014, pp. 27–36.

[87] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A Case for Stateful Forwarding Plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, Apr. 2013.

[88] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *Computer Communication Review*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[89] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named Data Networking (NDN) Project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, vol. 157, p. 158, 2010.

[90] Z. Zhang, Y. Yu, A. Afanasyev, J. Burke, and L. Zhang, "NAC: Name-based Access Control in Named Data Networking," in *Proc. of the 4th ACM Conference on Information-Centric Networking, (ICN)*, Berlin, Germany, Sep. 2017, pp. 186–187.

[91] F. Zhao, D. S. Lun, M. Médard, and E. Ahmed, "Decentralized Algorithms for Operating Coded Wireless Networks," in *Proc. of the IEEE Information Theory Workshop*, 2007, pp. 472–477.

[92] Y. L. S. Zhu and M. T. D.-Z. Du, "Localized Construction of Connected Dominating Set in Wireless Networks," in *Proc. of the US National Science Foundation International Workshop Theoritical Aspects of Wireless Ad Hoc, Sensor and Peer-to-Peer Networks*, 2004.

Declaration of consent

on the basis of Article 18 of the PromR Phil.-nat. 19

Name/First Name:

Registration Number:

Study program:

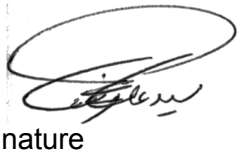
Bachelor ☐ Master ☐ Dissertation ☐

Title of the thesis:

Supervisor:

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 paragraph 1 litera r of the University Act of September 5th, 1996 and Article 69 of the University Statute of June 7th, 2011 is authorized to revoke the doctoral degree awarded on the basis of this thesis.
For the purposes of evaluation and verification of compliance with the declaration of originality and the regulations governing plagiarism, I hereby grant the University of Bern the right to process my personal data and to perform the acts of use this requires, in particular, to reproduce the written thesis and to store it permanently in a database, and to use said database, or to make said database available, to enable comparison with theses submitted by others.

Place/Date



Signature

Ali Marandi (CV)

Academic email: ali.marandi@inf.unibe.ch
Personal email: marandi.ali62@gmail.com

Education

PhD in Computer Science *2015-2020*
Institute of Informatics, the University of Bern, Switzerland

Dissertation title :
Bloom Filter-based Content Discovery and Retrieval in Information-Centric Networks

Research engineer *2013-2015*
Université Paris-Est, France

Task :
Implementing data collection protocol for large-scale Wireless Sensor Networks

M.Sc. Computer Engineering *2012*
Azad University, Iran

Dissertation title :
Efficient Information Dissemination in Delay Tolerant Networks

B.Sc. Computer Engineering, hardware design. *2009*
Azad University, Iran

Publications in this Thesis

A. Marandi, T.Braun, K. Salamatian, and N.Thomos,"**Network Coding-based Content Retrieval based on Bloom Filter-based Content Discovery for NDN**, in Proc. of the 54th IEEE Conference on Communications, 2020, Dublin, Ireland, Jun. 2020, pp. 1-7.

A. Marandi, T.Braun, K. Salamatian, and N.Thomos,"**Bloom Filter-based Routing for Dominating Set-based Service-Centric Networks**, in Proc. of the IEEE/IFIP Network Operations and Management Symposium, 2020, Budapest, Hungary, Apr. 2020, pp. 1-9.

M.Gasparyan, E.Schiller, A.Marandi, and T.Braun "**Communication Mechanisms for Service-Centric Networking**, in Proc. of the 17th IEEE Consumer Communications and Networking Conference, Las Vegas, NV, USA, Jan. 2020, pp. 1-9.

M.Gasparyan, A.Marandi, T.Braun, and E.Schiller, "**Fault-Tolerant Session Support for Service-Centric Networking**, in Proc. of the 17th IFIP/IEEE International Symposium on Integrated Network , Arlington, VA, USA, Apr. 2019, pp. 1-9.

A. Marandi, T.Braun, K. Salamatian, and N.Thomos, "**Pull-based Bloom Filter-based Routing for Information -Centric Networks**, in Proc. of the 16th IEEE Consumer Communications and Networking Conference, Las Vegas, NV, USA, Jan. 2019, pp. 1-7.

A. Marandi, T.Braun, K. Salamatian, and N.Thomos, "**A Comparative Analysis of Bloom Filter-based Routing Protocols for Information-Centric Networks**, in Proc. of the 23th IEEE Symposium on Computers and Communications, Natal, Brazil, Jun. 2018, pp. 1-7.

A. Marandi, T.Braun, K. Salamatian, and N.Thomos, "**BFR: a Bloom Filter-based Routing Approach for Information-Centric Networks**, in Proc. of the 16th International IFIP Networking Conference, Stockholm, Sweden, Jun. 2017, pp. 1-9.

Awards

My CCNC 2019 paper titled "Pull-based Bloom Filter-based Routing for Information-Centric Networks" was next to the runner-up paper and was the best of the track. The paper is available at `https://arxiv.org/abs/1809.10948`